

Security overview

Identity, encryption, tenant isolation, application security, platform monitoring, and supply-chain posture — the same content published at [/trust/security](#).

Identity and authentication

CMDash user accounts authenticate against a bcrypt-hashed password store (cost factor 12). The password policy has an 8-character minimum and a 72-character maximum on register and reset, and no composition rules by default; an organization can raise its own minimum length and add composition rules for password changes. CMDash does not check passwords against a breached-password list today. Passwords are never written to logs.

Time-based one-time password (TOTP, RFC 6238) multi-factor authentication is supported for every account. Recovery codes are stored as SHA-256 hashes, single-use. Per-tenant MFA enforcement policy supports "required for admins only" and a grace period so existing users have time to roll out second factors after policy is changed.

Logins are timing-equalised: the unknown-email path runs a bcrypt-compare against a per-process dummy hash so attackers cannot distinguish "user does not exist" from "wrong password" via response-time side channels. A regression test pins this.

Sessions are JWT access tokens with a 15-minute lifetime, paired with rotating refresh tokens (30-day default, 90-day opt-in for "keep me signed in"). Refresh tokens are stored as SHA-256 hashes in the database; a leaked refresh token is single-use because rotation returns a new one and invalidates the presenter. Refresh tokens are revoked on password reset, MFA disable, explicit logout, and admin-initiated "revoke all sessions".

Enterprise tenants can federate via SAML 2.0 (any IdP that speaks SP-initiated SSO) and provision users with SCIM v2. LDAP directory sync is available for on-premise deployments.

Authorisation and tenant isolation

Every request carries a tenant ID derived from the authenticated session. The Postgres role used by the API has row-level security (RLS) policies on every tenant-scoped table: a query cannot read rows outside the session's tenant, even if an application bug omits a WHERE clause. RLS is enforced at the database, not relying on application-layer filtering.

Role-based access control runs on three built-in roles (Owner, Admin, Viewer) plus per-tenant custom roles. Permission checks are guard-based, applied uniformly via decorators on every controller handler. Audit log rows capture the acting user ID on every write.

Encryption in transit and at rest

All HTTP traffic is served over TLS, with Strict-Transport-Security (max-age=31536000, includeSubDomains) on the application hosts. The header also requests preload, but neither cmdash.com nor cmdash.net is on the browser preload list yet; submission to the list is pending. Our load balancer's TLS policy currently accepts older protocol versions than we want it to; we are tightening it, and we are deliberately not publishing a minimum version on this page until that work lands. The certificate chain is a Lightsail load-balancer certificate in cloud deployments; on-prem deployments manage their own chain through the operator-supplied Caddy config.

At rest, object storage (S3) uses server-side AES-256 encryption with S3-managed keys. Our production database is a Lightsail managed PostgreSQL instance; this repository's infrastructure code does not document its at-rest encryption configuration, so we are not asserting a specific mechanism here. New evidence attachments, evidence-package archives and data exports are sealed with AES-256-GCM under a per-tenant data key before they are written to object

storage, and files stored before that change are converted by a one-time migration. Each tenant's key is stored only in wrapped form, under the platform master key or an AWS KMS key where one is configured, and every sealed object is bound to its tenant and its storage path. When a workspace is offboarded its data keys are destroyed first, which makes its sealed objects permanently unreadable, and the files and rows are then deleted. Other files, and the database, are not sealed with per-tenant keys. A database backup taken before a key was destroyed still holds that key in wrapped form until the backup ages out.

Transport security and HTTP hardening

The dashboard, the marketing site and the API apply a strict set of HTTP response headers at two independent layers — in the web applications themselves (Next.js) and in the API (helmet in NestJS). The set includes:

- Strict-Transport-Security with one-year max-age and includeSubDomains, plus a preload request (see above);

- X-Frame-Options: SAMEORIGIN alongside a CSP frame-ancestors 'self' directive — no third-party site may frame the dashboard, while the dashboard's own pages (the landscape atlas, the embedded API reference) may. The marketing site, the staff portal and the policy portal frame nothing and are set to DENY;

- X-Content-Type-Options: nosniff, Referrer-Policy: strict-origin-when-cross-origin ;

- Cross-Origin-Opener-Policy and Cross-Origin-Resource-Policy set to same-origin;

- Permissions-Policy denying camera, microphone, and geolocation;

- A nonce-based Content-Security-Policy, minted per request, with no unsafe-inline or unsafe-eval in script-src on the dashboard, the staff application and the marketing site. It is enforcing on the dashboard. On the public marketing site it currently runs in report-only mode pending soak evidence, so violations there are recorded rather than blocked.

One application does not yet meet that set: the policy portal sets the transport, framing, isolation and permissions headers but no Content-Security-Policy of its own. The staff administration application mints a per-request nonce and carries no `unsafe-inline` or `unsafe-eval` in `script-src`. The header values of the dashboard, the marketing site, the staff application, the policy portal and the API are pinned by tests, so a refactor cannot silently regress them.

Corrected 2026-09-07. This page previously described the two layers as “the edge (Nginx or CloudFront)” and the application. That was accurate for an earlier deployment topology and stopped being true when the edge moved to Caddy: for a period, the six non-CSP headers above reached API responses but not the HTML served by the web applications, and no test asserted them. Both halves are fixed — the applications set the headers themselves, and the tests referenced above now exist. We would rather show the correction than quietly restate the claim.

Application security controls

Input validation runs as a per-endpoint concern. Many controllers parse request bodies with zod schemas, and a structural body guard rejects malformed or oversized JSON before it reaches a handler, but not every write handler has a schema yet. Database access goes through Prisma, which generates parameterised queries. Raw SQL is used in about 40 files; every raw query is parameterised and none builds a statement by string interpolation of user input.

Write endpoints that represent idempotent business actions accept an Idempotency-Key header and return the same response on retry. Webhook receivers verify an HMAC signature against a shared secret with a constant-time compare, and bound replay windows through a timestamp tolerance. Outbound webhook deliveries sign the body with the same HMAC scheme.

Asynchronous jobs that fail repeatedly move to a dead-letter queue with a retry budget, so a broken dependency cannot spin an infinite retry loop. Outbound integrations sit behind a circuit breaker so a degraded partner cannot cascade into a CMDash outage. Feature flags gate risky changes for progressive rollout and instant rollback.

Logging, monitoring, and SLOs

Every request logs through a structured logger (pino) with a consistent shape: request ID, tenant ID, user ID, path, status, latency. Trace context (W3C traceparent) propagates from the edge through to downstream jobs and webhooks. Unhandled exceptions are logged with their stack and request context to the same structured log.

Prometheus scrapes API, database, Redis and host metrics (latency, error rate, queue depth, database connections, cache saturation) on both cloud and on-prem deployments. Service-level objectives for API availability and latency are published; see the SLO definitions linked from the SOC 2 page. Automated alerting on those metrics is written but not yet applied to production — today an operator has to notice a dashboard rather than receive a page. That includes the cloud deployment's CloudWatch alarms and SNS paging topic and burn-rate paging against the published SLOs.

Security-relevant events — login, MFA change, password reset, session revoke, role change, tenant create/delete — are written to the audit log. A tamper-evident hash chain over that log is built but not switched on in production. The default retention is 365 days: a daily purge at 02:00 UTC deletes older entries for each organization that has saved a retention policy. CMDash's own operational tables (sign-in session records, health checks, the e-mail send record, billing de-duplication markers, marketing leads) are swept every night on fixed windows listed in the privacy policy, and e-mail addresses are masked before an application log line is written. Automated anomaly detection over that log (for example, impossible travel or burst auth failures) is not built yet; today it is reviewed manually.

Secrets and credential handling

Production secrets are held in an environment file on each host, outside images and source control. AWS Secrets Manager holds the database credential and the JWT signing secret only; on-prem deployments keep theirs in the operator's own secret store. The repository is scanned with gitleaks in the local pre-push and release gate, and a detection fails that gate; it is not run by a hosted CI service on

every push. The secret-leak response runbook includes rotation instructions for every secret class we hold.

Operator access to production follows principle-of-least-privilege: break-glass paths require MFA and write to the audit log with a reviewer signoff in the runbook.

Supply chain

Every released container image (published to GitHub Container Registry) is scanned and then signed by digest with Cosign, using a release key pair. The private key and its passphrase stay on the release host; the public key is committed in the repository at `scripts/security/cosign-release.pub`, and consumers verify an image by digest against it. Because this is a long-lived key rather than a keyless identity, key rotation is our responsibility. See `docs/security/supply-chain.md`.

Each image ships with a CycloneDX Software Bill of Materials, attached as a Cosign attestation. When a CVE lands, we pull the SBOM, grep for the vulnerable package, and either ship a dependency bump or (future) publish a VEX advisory explaining why the usage is not exploitable. Automated dependency update proposals keep the graph current.

Build provenance (SLSA Level 3) and signature-verification gating at deploy time are open hardening items on the supply-chain roadmap.

Availability and recovery

The cloud deployment runs on a Lightsail managed PostgreSQL database with automated daily snapshots and point-in-time recovery; the module provisions it on a high-availability bundle by default. There is no cross-region backup replication and no secondary region today. The application tier runs multiple Lightsail instances, distributed across availability zones, behind a Lightsail load balancer with health-checked rolling deployments. There is no CDN caching application or API responses.

A documented monthly restore drill — exercising the full backup-to-running-service path — is an open item on the availability roadmap. Backups are verified for retrievability today; the end-to-end drill cadence lands during the current compliance wave.

Verify it yourself

Read the live Trust Center, or start a trial.

Every claim above corresponds to shipped code or a linked policy — nothing is aspirational.