FREE DOWNLOAD · MIT LICENSED

Cloud Controls Evidence Kit

A drop-in scaffold for B2B SaaS engineering teams responding to customer security reviews, SOC 2 readiness work, and compliance-platform checks.

COMPILED BY

Musah Abdulai

Cloud controls & security-review implementation

Google Cloud Professional DevOps Engineer

May 21, 2026 · musabdulai.com/evidence-kit

Top level

README.md

Cloud Controls Evidence Kit

A drop-in scaffold for B2B SaaS and AI-product teams to organize the engineering controls and evidence that customer security reviews, SOC 2 readiness work, and compliance platforms (Vanta, Drata, Secureframe) ask for. Markdown source, MIT licensed, free to fork, edit, and use.

What's inside

Folder / file	What it is
`controls-map.md`	Master mapping: control → required evidence → where to find it → owner → update cadence. ~30 controls across 6 categories.
`questionnaire-answer-examples.md`	12 template answers for the questions that come up on customer security questionnaires (MFA, audit logs, backups, etc.).
`evidence-folder-template/`	Ready-to-use folder layout — six numbered category folders, each with a README and 3–4 evidence templates. Drop into your evidence repo and start filling in.
`platforms/aws.md`, `gcp.md`, `azure.md`, `github.md`	Where to find each piece of evidence on each platform: console paths, CLI commands, exports that satisfy auditors.
`LICENSE`	MIT. Use it, edit it, ship it, redistribute it.

How to use it

1. **Fork or download** this kit (link on the evidence kit page on musabdulai.com).

2. **Drop `evidence-folder-template/` into your own private repo** and rename it to whatever your team calls evidence. Many teams use `compliance/` or `audit/`.
3. **Open `controls-map.md`** and adapt the table to your actual systems. Delete rows that don't apply, add rows for systems we don't cover.
4. **Fill in each template** as you collect the evidence. Each template tells you what goes there, where to find it, and how often to refresh it.
5. **When a customer questionnaire arrives**, paste answers from `questionnaire-answer-examples.md` and attach the matching evidence from your folder.

Tone and scope

This kit is opinionated and concrete. It names specific tools (CloudTrail, Vanta, GitHub branch protection, Vertex AI) instead of staying framework-agnostic — because that's how the controls actually map at most SaaS shops.

It does **not** cover:

- Drafting policies (privacy policy, security policy text) — talk to a lawyer; tools like StrongDM's Comply are good for policy drafting.
- Compliance-platform-specific configuration (each platform has its own docs).
- Threat modeling, red-team plans, or incident response procedures — different scope.

This kit covers **the operational evidence engineering teams have to produce** when a buyer or auditor asks for proof that a control is real.

Updating cadence

Customer security reviews and auditors look for evidence that is **fresh**. The rule of thumb on cadence:

Evidence type	Refresh
Policy / config snapshots (MFA policy, branch protection, retention)	Quarterly
Operational logs / exports (IAM key inventory, deploy history)	Quarterly
Restore tests, DR drills	Quarterly or semi-annually
Incident runbook, on-call rotation	Annually unless a real incident
Vendor reviews, third-party access	Annually

Each evidence template includes a recommended cadence.

License

MIT. Use it however helps you ship.

Maintained by

Musah Abdulai · cloud controls implementation for B2B SaaS and AI-product teams · musabdulai.com · hello@musabdulai.com

If this kit saved you a day of evidence-gathering work and you'd like an engineer to actually do the work for you, the website has a sample report showing what a Controls Review deliverable looks like: musabdulai.com/sample-report.

controls-map.md

Controls Map

The 30 controls that come up on the vast majority of customer security questionnaires, SOC 2 readiness assessments, and compliance-platform checks. Use this as the index of your evidence folder.

For each control: the evidence required, where it lives, who owns it, and how often to refresh.

> **Tip:** delete rows for controls that don't apply (e.g. AI workload rows if you ship no AI features), and add rows for systems unique to your stack. The structure matters more than the exact list.

1. Access controls

#	Control	Evidence	Where to find	Owner	Cadence
1.1	MFA enforced for all admin users	IdP MFA policy export + per-user enforcement screenshot	Okta / Google Workspace / Microsoft Entra → Policies	IT / Security	Quarterly
1.2	Least-privilege IAM roles	IAM role definitions + per-role member list	AWS IAM / GCP IAM / Azure AAD	Platform	Quarterly
1.3	Service-account / long-lived key inventory	Inventory + rotation policy + last-rotated timestamps	AWS IAM access keys / GCP IAM service-account keys / GitHub PATs	Platform	Quarterly

#	Control	Evidence	Where to find	Owner	Cadence
1.4	Offboarding revokes all access	Offboarding checklist + 3–5 completed runs	Internal runbook + IdP audit log + Cloud audit logs	HR / IT	Per offboarding
1.5	Privileged access review	Quarterly review minutes + sign-off	Internal doc + IdP membership export	Security	Quarterly

2. Logging and monitoring

#	Control	Evidence	Where to find	Owner	Cadence
2.1	Cloud audit logging enabled across all production accounts	Trail / log-sink config export covering every prod account or project	AWS CloudTrail Org Trail / GCP Cloud Audit Logs sink / Azure Activity Log	Platform	Quarterly
2.2	Audit log retention ≥ 365 days for the review period	Storage lifecycle policy + sample query showing oldest available log	S3 / GCS / Storage Account lifecycle rules	Platform	Annually
2.3	Application logs collected centrally	Logging config + sample query	Datadog / Honeycomb / Grafana Loki / native	Platform	Annually
2.4	Security alerts route to on-call	Alert configuration + sample firing	PagerDuty / Opsgenie / native cloud alerts	Security	Annually

#	Control	Evidence	Where to find	Owner	Cadence
2.5	Tamper-resistant log storage	Object-lock / WORM / immutability config	S3 Object Lock / GCS Bucket Lock	Platform	Annually

3. CI/CD and change management

#	Control	Evidence	Where to find	Owner	Cadence
3.1	Required reviews on production deploy	Branch-protection config + CODEOWNERS	GitHub / GitLab branch rules	Engineering leads	Quarterly
3.2	Environment-protected production deploys	GitHub Environments / GitLab protected environments config	Repo → Settings → Environments	Engineering leads	Quarterly
3.3	Signed / verified deploy artifacts	Sigstore / SLSA attestation samples (when adopted)	CI build logs	Platform	Quarterly
3.4	Release evidence: who shipped what, when	Last 30 production deploys with author + approval trail	CI/CD history + PR merges	Engineering leads	Per deploy / quarterly
3.5	Rollback capability documented	Runbook + last rollback exercise	Internal docs	Platform	Annually

4. Vulnerability and secrets hygiene

#	Control	Evidence	Where to find	Owner	Cadence
4.1	Dependency scanning enabled with SLA on critical	Scanner config (Dependabot / Renovate / Snyk) + remediation log	Repo settings + security advisories	Platform	Quarterly
4.2	Container / image scanning	Scanner config + last 30 deploys' scan results	Trivy / Artifact Registry / ECR scanning	Platform	Quarterly
4.3	Secret scanning enabled, blocking on detection	GitHub Advanced Security / GitLab secret detection config + remediation log	Repo security tab	Security	Quarterly
4.4	Pen-test or red-team within 12 months	Engagement report or summary	Internal doc / vendor portal	Security	Annually

5. Backups, recovery, and availability

#	Control	Evidence	Where to find	Owner	Cadence
5.1	Production databases backed up	Backup configuration + last 7 backup success timestamps	RDS / Cloud SQL / native	Platform / SRE	Quarterly
5.2	Backup restore tested	Restore-test runbook + last test	Internal doc	Platform / SRE	Quarterly

#	Control	Evidence	Where to find	Owner	Cadence
		report (row counts, runtime)			
5.3	Incident runbook exists	Runbook + last incident's postmortem	Internal docs / Notion	Engineering leads	Annually
5.4	Status page or customer-facing availability signal	Public status page URL + incident history	StatusPage / Atlassian / native	Platform	Annually

6. AI workload controls

#	Control	Evidence	Where to find	Owner	Cadence
6.1	Model endpoints behind auth	Auth config (API gateway / VPC / IAM) + sample request denied without token	Vertex AI / OpenAI org settings / Azure OpenAI	AI / Platform	Quarterly
6.2	Prompt / tool-call logging policy	Policy doc covering what's logged, retention, and access	Internal doc	AI / Security	Annually
6.3	Per-tenant or per-org spend caps	Quota / rate-limit config + spend alert routing	Vertex AI quotas / OpenAI org limits / custom rate-limiter	AI / Platform	Quarterly

#	Control	Evidence	Where to find	Owner	Cadence
6.4	RAG document access boundaries	Tenant-isolation policy + index/namespace mapping	Internal doc + vector DB config	AI	Quarterly
6.5	AI data retention reviewed	Retention policy aligned to customer contracts + DPA terms	Internal doc	AI / Legal	Annually
6.6	Abuse / misuse handling	Runbook for prompt-injection reports, content abuse, leaked-prompt incidents	Internal doc	AI / Security	Annually

How to use this table

1. **For each row that applies to your stack**, create a corresponding evidence file in your `evidence-folder-template/<category>/` folder. Use the templates in this kit as starting points.
2. **For each customer security questionnaire**, scan the questions and pull from this table to find the right evidence to attach. The `questionnaire-answer-examples.md` file has template answers that reference the standard evidence categories.
3. **At the quarterly review cadence**, walk the table and refresh anything overdue. Many controls drift quietly (key rotation, deploy approver lists, MFA exceptions); a quarterly walk catches them.

Mapping to common frameworks

This kit	SOC 2 Trust Service Criterion	NIST CSF
1.x Access controls	CC6.1, CC6.2, CC6.3	PR.AC
2.x Logging and monitoring	CC7.2, CC7.3	DE.CM, DE.AE
3.x CI/CD and change management	CC8.1	PR.IP-3
4.x Vulnerability and secrets hygiene	CC7.1	PR.IP-12, DE.CM-8
5.x Backups, recovery, availability	CC9.1, A1.2	RC.RP, PR.IP-4
6.x AI workload controls	(emerging — currently scoped under CC6 / CC7)	(emerging — see NIST AI RMF)

These mappings are approximate; your auditor or compliance platform will have the authoritative cross-walk.

---

questionnaire-answer-examples.md

Questionnaire Answer Examples

Template answers for the 12 questions that come up on almost every customer security questionnaire. Paste, edit to fit your specifics, attach the matching evidence from your `evidence-folder-template/`.

> **House rules:**
>
> 1. Don't bluff. If a control doesn't exist, write "We do not currently enforce this; we plan to by [date]" instead of inventing one.
> 2. Always attach the evidence. Answer-only questionnaire responses get follow-up requests; evidence-attached responses close the loop.
> 3. Keep answers short. Procurement teams skim. The evidence file is where the detail lives.

Access controls

1. Do you enforce MFA for all administrative access to production systems?

> Yes. MFA is enforced via {{IdP — e.g. Okta / Google Workspace / Microsoft Entra}} for all users with administrative roles in our identity provider, cloud consoles, and code-repository organization. Enforcement is policy-based at the IdP, not opt-in. We review the per-user enforcement status quarterly.

> **Evidence:** see `01-access-controls/mfa-enforcement.md` and the attached IdP policy export.

2. How do you manage privileged access?

> Privileged access is granted via named roles in our IdP (no shared credentials, no permanent admin assignments to individual users where avoidable). Membership in privileged roles is reviewed quarterly and recorded in `01-access-controls/iam-quarterly-review.md`. Production infrastructure access for engineers is granted via {{your access system — e.g. AWS SSO / GCP IAM groups / Teleport / Tailscale}}.

3. How do you handle offboarding?

> Within {{N hours}} of a departure, the offboarding checklist revokes access across the IdP, all cloud providers, all code repositories, and SaaS tools. The completed checklist is filed in `01-access-controls/offboarding-checklist.md` per departure. Quarterly we sample 3 recent offboardings and verify no orphaned credentials remain.

Logging and monitoring

4. How long do you retain audit logs?

> Production cloud audit logs are retained for {{N}} days in {{S3 / GCS / Storage Account}} with object-lock / bucket-lock applied to prevent tampering. Application logs are retained for {{N}} days in {{Datadog / Honeycomb / Loki / native}}.
>
> **Evidence:** see `02-logging-monitoring/retention-policy.md` for the storage lifecycle configuration and a sample query showing the oldest available log.

5. Do you have intrusion detection / security monitoring?

> Yes. Security-relevant events from cloud audit logs, IdP audit logs, and code-repository security alerts route to our on-call rotation via {{PagerDuty / Opsgenie}}. Alert rules and routing are documented in `02-logging-monitoring/alert-routing.md`. We do not operate a dedicated SOC; alerts are triaged by the on-call engineer against the runbook in `05-backups-recovery/incident-runbook.md`.

CI/CD and change management

6. How do you control changes to production?

> Production deploys are gated by:
>
> 1. Branch protection on the production branch ({{main / production}}) — at least one required PR review by a CODEOWNER.
> 2. GitHub Environments protection on the production environment — requires explicit approval by an authorized reviewer.

3. Required passing CI checks (lint, type-check, tests, security scans).

Evidence: see `03-cicd-change-management/branch-protection.md` (rule export), `deploy-approvals.md` (environment config), and `release-evidence.md` (last 30 production deploys with author and approver).

7. Can you produce a change history for production?

Yes. Every production deploy is associated with a Pull Request, CODEOWNER review, and CI build. The full history is available via `git log` on the production branch and our CI/CD platform's deploy history. The most recent 30 deploys are exported quarterly to `03-cicd-change-management/release-evidence.md`.

Vulnerability and secrets hygiene

8. Do you scan for vulnerabilities and secrets?

Yes:

- **Dependencies:** {{Dependabot / Renovate / Snyk}} scans every pull request and continuously against `main`. Critical/high CVEs are remediated per the SLA in `04-vulnerability-secrets/dependency-scanning.md`.
- **Container images:** {{Trivy / Artifact Registry / ECR}} scanning runs in CI; deploys are blocked on critical CVEs unless explicitly exempted via the documented exception process.

- **Secrets in repositories:** GitHub Advanced Security secret scanning is enabled organization-wide with push protection. Hits are triaged within {{N hours}} per `04-vulnerability-secrets/secret-scanning.md`.

9. Have you had a penetration test?

{{Yes — vendor and date}}, or:

> We have not engaged a third-party pen-test as of {{date}}. We plan to in {{quarter / year}}. Our current security posture is validated through automated scanning (dependency, container, secret), code review on every change, and {{any in-house red-team exercises}}.

Don't fake a pen-test you didn't do.

Backups, recovery, and availability

10. How are production databases backed up, and have you tested restore?

{{RDS / Cloud SQL / native}} automated backups run {{daily / continuously}} with {{N}}-day retention. Point-in-time recovery is enabled. Restore tests are run quarterly to a non-production environment; the most recent test result (date, restored row counts, wall-clock time, integrity-check outcome) is in `05-backups-recovery/restore-test.md`.

11. What's your RPO / RTO?

> Recovery Point Objective (RPO): {{N minutes / hours}}, established by the continuous backup capability above.
>
> Recovery Time Objective (RTO): {{N hours}}, validated by the quarterly restore test which measures wall-clock time from snapshot selection to fully populated target instance.
>
> The incident runbook in `05-backups-recovery/incident-runbook.md` includes a documented major-incident recovery procedure.

AI workload (if applicable)

12. How do you handle AI / LLM-specific security and abuse?

> Our AI features operate under documented controls:
>
> - **Endpoint auth:** all model inference endpoints sit behind per-tenant API authentication. Public unauthenticated access is not possible.
> - **Prompt and tool-call logging:** logged centrally per the policy in `06-ai-workload-controls/prompt-tool-call-logging.md`, with retention aligned to our customer DPA.
> - **Tenant isolation:** retrieval indices, embeddings, and any stored AI artifacts are partitioned per-tenant. See `06-ai-workload-controls/data-retention.md`.
> - **Spend / abuse controls:** per-tenant rate limits and spend caps are enforced in front of the model endpoints; alerts route to on-call. See `06-ai-workload-controls/spend-caps.md`.

> - **Customer data and training:** we do not use customer data to train or fine-tune base models. {{Adjust if not true.}}

When the questionnaire asks something not on this list

Common follow-ups:

Topic	Where to look
Vendor / sub-processor list	Internal vendor inventory
Data residency	Cloud region selection + storage policy
Encryption at rest / in transit	Cloud-provider defaults + custom KMS config
Employee security training	HR/training records
Background checks	HR records
SOC 2 / ISO report	Engage an auditor — this kit does not substitute for one

If a question genuinely doesn't map to anything you have, **don't invent the control**. Write the honest version: "We do not currently enforce this. We assess the risk as {{LOW / MEDIUM / HIGH}}. We plan to implement this by {{date}}." That answer is fine for many buyers; it's the bluff that gets caught.

Evidence folder template

evidence-folder-template/README.md

Evidence folder template

Drop this directory structure into your private compliance/evidence repo and adapt it to your stack. Each numbered folder maps to a control category in `../controls-map.md`.

Structure

```
evidence-folder-template/
  01-access-controls/
  02-logging-monitoring/
  03-cicd-change-management/
  04-vulnerability-secrets/
  05-backups-recovery/
  06-ai-workload-controls/
```

Each category folder contains:

- A `README.md` explaining what evidence belongs there and who owns it.
- Three to four `*.md` templates for the most common evidence files.
- Supporting attachments (screenshots, exports, CSVs) when you add them.

Naming conventions

When a customer security review starts pulling files, name them so the procurement team doesn't have to ask follow-ups:

Naming pattern	Example	Why
`<topic>-<date>.png`	`mfa-policy-2026-Q2.png`	Screenshot of a console policy

Naming pattern	Example	Why
`<topic>-<period>.md`	`quarterly-iam-review-2026Q1.md`	Periodic review minutes
`<topic>-config.{json,yaml}`	`cloudtrail-org-config.json`	Configuration export
`<topic>-inventory.csv`	`service-account-keys-2026-05.csv`	Inventory export

Date format: ISO-8601 (`2026-05-21`) or quarter (`2026Q2`). Don't use American or European date formats — they're ambiguous globally.

Refresh cadence

Each evidence file template includes a recommended cadence at the top. The default rule of thumb:

Type	Refresh
Policy configuration (MFA, branch protection, retention)	Quarterly
Operational logs / exports (deploy history, IAM inventory)	Quarterly
Restore tests, DR drills	Quarterly
Vendor / third-party access reviews	Annually
Incident runbook, on-call rotation	Annually unless a real incident

Set a recurring calendar event for the quarterly review. Walk the folder, refresh anything older than 90 days, archive what's no longer relevant.

Privacy

This template assumes the evidence folder lives in a **private** repository or storage. Some evidence files (IAM exports, key inventories, deploy histories with author names) contain identifiers. Don't make this folder public.

If you need to share evidence with a customer or auditor, copy the specific files into a shared evidence portal (Vanta Trust Center, Drata Trust Page, Conveyor, OneTrust, or your own shared drive) — don't grant access to the full evidence repo.

What this folder is NOT

- A substitute for a SOC 2 audit. An auditor still needs to inspect controls and issue a report. This folder makes that inspection efficient; it doesn't replace it.
- A policy library. Policies (acceptable-use, security, privacy) live in your policy repo. See StrongDM's Comply for a markdown-based policy template framework.
- A compliance platform. Vanta / Drata / Secureframe pull evidence automatically from cloud APIs; this folder is the manual complement (and the source of truth when automation misses something).

Access controls

evidence-folder-template/01-access-controls/README.md

Access controls

Evidence that the people and machines accessing production systems have appropriate, current authentication and authorization.

What goes here

- MFA enforcement evidence (IdP policy + per-user status)
- Quarterly IAM review minutes (who has admin, who shouldn't)
- Service-account / long-lived key inventory with rotation status
- Completed offboarding checklists (sample of 3-5 recent)

Owner

Typically split: **IT / Security** owns the IdP layer; **Platform** owns the cloud IAM layer; **HR + IT** jointly own offboarding.

Common gotchas

- **Long-lived service-account keys.** Almost every audit catches at least one of these. Inventory + rotation policy + last-rotated date is the standard evidence. Migration to Workload Identity (GCP) or IAM Roles for Service Accounts (AWS EKS) is the long-term fix.
- **GitHub Personal Access Tokens with org admin.** Common blind spot. Audit org members' PATs and replace with GitHub Apps where possible.
- **MFA exceptions for "break-glass" accounts.** Document them, scope the credential storage, rotate quarterly. Don't pretend they don't exist.
- **Standing admin access for engineers.** Replace with just-in-time elevation via {{Teleport / AWS SSO / GCP IAM Conditions}} where possible. If not, document who and why.

Cross-references

- Controls map: rows **1.1 – 1.5** in `../../controls-map.md`
- Platform guides: see `../../platforms/aws.md`, `gcp.md`, `azure.md`, `github.md` for where to find each piece of evidence per platform
- Questionnaire answers: questions 1-3 in `../../questionnaire-answer-examples.md`

`evidence-folder-template/01-access-controls/iam-quarterly-review.md`

IAM quarterly review

Periodic review of who has privileged access — to the IdP, cloud consoles, code repositories, and production data. Catches drift: former contractors still in groups, engineers who got temporary elevation and never lost it, service accounts that aren't used anymore.

Evidence to keep here

1. **Review minutes** for each quarter — date, attendees, decisions made (who got removed, who got added, who got justified to stay).
2. **Membership export** at the time of review — IdP admin role members, cloud-console privileged roles, repo org owners / admins, key SaaS admin accounts.
3. **Sign-off** — name(s) of the reviewer who approved the resulting membership list.

How to gather it

Quarterly:

1. Export current membership:
   - IdP: Okta / Google Workspace / Entra → admin role membership
   - AWS: `aws iam get-account-authorization-details` or AWS Identity Center assignments
   - GCP: `gcloud projects get-iam-policy <project>` for each prod project
   - GitHub: org → People → filter by `2fa_disabled` and `owners` role
   - SaaS critical tools: Snowflake admin users, Datadog admin users, etc.
2. Walk each list with the responsible owner. For each member, answer: "Is this person still in a role that requires this access?" Remove anyone who's a no.
3. Write up the diff: removed, added, kept-with-justification.
4. Both reviewers sign off (file in this folder).

Template review minutes

```
# IAM Review — 2026Q2 (2026-05-15)

**Reviewers:** {{name}} (Security), {{name}} (Platform)

## Removed

| System | Role                          | Member    | Reason                 |
| ------ | ----------------------------- | --------- | ---------------------- |
| AWS    | OrganizationAccountAccessRole | alice@... | Left company 2026-04-12 |
| GitHub | Owner                         | bob@...   | Moved to non-admin role |

## Added

| System   | Role        | Member    | Reason                    |
| -------- | ----------- | --------- | ------------------------- |
| GCP prod | roles/owner | carol@... | Promoted to Platform Lead |

## Kept (with justification)

| System | Role                | Member   | Justification                          |
| ------ | ------------------- | -------- | -------------------------------------- |
| AWS    | AdministratorAccess | dave@... | Sole on-call admin for prod break-glass |

## Sign-off

- Reviewed by {{name}}, {{name}} on 2026-05-15.
- Next review: 2026-08-15.
```

Example filenames

```
quarterly-iam-review-2026Q2.md
iam-membership-snapshot-2026-05-15.csv
```

Refresh

Quarterly. Set a recurring calendar event with both reviewers.

evidence-folder-template/01-access-controls/mfa-enforcement.md

MFA enforcement

The single most-requested control on customer security questionnaires. Buyers want to see that **every** human with admin access uses MFA, and that the enforcement is policy-level (not opt-in).

Evidence to keep here

1. **IdP MFA policy export** — the actual rule, not a screenshot of the policy-editor UI. JSON, YAML, or PDF depending on what your IdP exports.
2. **Per-user enforcement screenshot or report** — showing all admin users with an MFA status of "Enabled" (or equivalent). Include the date in the filename.
3. **Exception register** — if any accounts skip MFA (break-glass / service / legacy), list them with justification and compensating controls. Don't omit them; auditors find them.

How to gather it

- **Okta**: Security → Authentication → Authentication Policies → export the policy. Also Reports → Users → MFA Usage.
- **Google Workspace**: Admin → Security → 2-Step Verification → Enforcement settings + Reports → Audit → Login (filter for `is_2sv = false` to find any holes).
- **Microsoft Entra (Azure AD)**: Entra ID → Security → Conditional Access → the policy enforcing MFA on the relevant role/group. Reports → Sign-in logs filtered by MFA result.

Common buyer questions answered

> "Are all administrators MFA-enabled?"

Yes — see attached `mfa-policy-<date>.{json,pdf}` and `mfa-status-<date>.csv`.

> "Are there exceptions?"

{{Pick one:}}

- No exceptions.
- {{N}} exceptions: {{break-glass account / monitoring service}}. Justification in `mfa-exceptions-<date>.md`. Compensating controls: {{credential vault, hardware token, IP allowlist, etc.}}.

Example filenames

```
mfa-policy-2026-05-15.json
mfa-status-export-2026-05-15.csv
mfa-exceptions-2026-05-15.md       # only if applicable
```

Refresh

Quarterly. Set a recurring calendar event. The check takes <10 minutes once you've done it once.

---

evidence-folder-template/01-access-controls/offboarding-checklist.md

Offboarding checklist

Auditors and customer security reviews ask for evidence that access is revoked promptly when someone leaves. The standard evidence is: (a) a documented checklist, (b) completed instances of recent offboardings, (c) a periodic spot-check that no orphaned access remains.

Evidence to keep here

1. **This checklist** — the authoritative procedure.
2. **3–5 most recent completed instances** — one file per departure, showing each step completed with timestamp.
3. **Quarterly orphan check** — a quick scan confirming none of the recently-departed have active credentials anywhere.

The checklist (master template)

Within **{{N hours}}** of HR signaling a departure, IT completes:

- ☐ Disable IdP account (Okta / Google / Entra) — wipes SSO access
- ☐ Revoke active sessions on IdP
- ☐ Remove from any admin / privileged role groups
- ☐ Cloud providers:
  - ☐ AWS: deactivate IAM user / remove SSO assignment
  - ☐ GCP: remove from organization IAM policies
  - ☐ Azure: remove from Entra groups + role assignments
- ☐ Code repositories:
  - ☐ GitHub: remove from org (or convert to outside collaborator if needed temporarily)
  - ☐ Revoke any active Personal Access Tokens issued to them

- ☐ SaaS criticals:
  - ☐ Datadog, Sentry, PagerDuty, Linear, Notion, Slack, etc.
- ☐ Production secrets:
  - ☐ If they had access to a vault (1Password / Vault / KMS), rotate any shared secrets they knew
- ☐ Hardware:
  - ☐ Laptop returned and wiped
  - ☐ YubiKey returned (or revoked from registration)
- ☐ Email forwarded / archived per data-retention policy
- ☐ Final attestation: `git log --author="<name>"` + IdP audit log show no activity after revocation timestamp

Per-instance template

For each offboarding, copy this template to `offboarding-<initials>-<YYYY-MM-DD>.md`:

```
# Offboarding: {{Initials}}, departed 2026-MM-DD

| Step                     | Done at          | By            |
| ------------------------ | ---------------- | ------------- |
| IdP disabled             | 2026-MM-DD HH:MM | {{IT person}} |
| Sessions revoked         | 2026-MM-DD HH:MM | {{IT person}} |
| AWS access removed       | 2026-MM-DD HH:MM | {{Platform}}  |
| GCP access removed       | 2026-MM-DD HH:MM | {{Platform}}  |
| GitHub access removed    | 2026-MM-DD HH:MM | {{Eng leads}} |
| PATs revoked (N=...)     | 2026-MM-DD HH:MM | {{Eng leads}} |
| SaaS tools               | 2026-MM-DD HH:MM | {{IT person}} |
| Shared secrets rotated   | 2026-MM-DD HH:MM | {{Platform}}  |
| Hardware returned/wiped  | 2026-MM-DD HH:MM | {{IT person}} |
| Final attestation        | 2026-MM-DD       | {{Reviewer}}  |

Notes: {{anything atypical, e.g. retained advisor relationship}}
```

Quarterly orphan check

Walk this list each quarter:

- IdP: list users not logged in for 90+ days → confirm each is intentional
- GitHub: outside collaborators list → confirm each is current
- AWS access keys / GCP service-account keys: any owned by a former employee → rotate immediately
- Datadog / Sentry / Linear / Slack: external/inactive users

File the result as `orphan-check-<YYYY-QN>.md`.

Refresh

Per departure (mandatory). Plus quarterly orphan check.

Logging & monitoring

evidence-folder-template/02-logging-monitoring/README.md

Logging and monitoring

Evidence that you can answer "what happened in production?" and "did anyone notice?" — with logs that are complete, retained long enough, and routed somewhere that responds.

What goes here

- Cloud audit log configuration (CloudTrail / Cloud Audit Logs / Azure Activity Log)
- Retention policy + lifecycle config
- Alert routing config (PagerDuty, Opsgenie, etc.) + sample fires
- Tamper-resistance config (object-lock / bucket-lock / WORM)

Owner

Typically **Platform** for log configuration and **Security** for alert content. Some shops merge these.

Common gotchas

- **Per-account audit trails instead of org-level.** A trail only in one account leaves other accounts un-logged. Org-level CloudTrail (or GCP Org-level audit log sink) is the only acceptable answer for multi-account orgs.
- **Logs in the same account as the workload.** If the workload is compromised, the logs are too. Store in a dedicated security / logging account or project with limited write access.
- **Retention "until disk fills."** SOC 2 expects you can pull the full review-period audit trail (typically 1 year). Explicit lifecycle policy is the evidence.
- **No tamper resistance.** Audit logs in mutable storage are worth less. S3 Object Lock, GCS Bucket Lock, or equivalent.

- **Alerts but no one paged.** "We have alerts" without a routing destination is a documentation-only control. Verify that alerts actually fire to a person (or runbook) within minutes.

Cross-references

- Controls map: rows **2.1 – 2.5** in `../../controls-map.md`
- Platform guides: see `../../platforms/` for where to find each piece of evidence per cloud
- Questionnaire answers: questions 4-5 in `../../questionnaire-answer-examples.md`

---

evidence-folder-template/02-logging-monitoring/alert-routing.md

Alert routing

Evidence that security-relevant events route to a human (or documented runbook) within minutes. "We have alerts configured" is not enough — the questionnaire wants proof of routing and acknowledgement.

Evidence to keep here

1. **Alert rules** — list of security-relevant alerts with their triggers, severities, and destinations.
2. **Routing config** — PagerDuty / Opsgenie / native cloud alert integration that shows messages reaching a person.
3. **Sample fires** — 1-3 anonymized examples of alerts that actually fired, were acknowledged, and resolved. Demonstrates the chain works.

Alerts that customer security reviews expect

At minimum:

Trigger	Severity	Destination
Root / break-glass account login	Critical	Page on-call
New IAM admin role assignment	High	Page on-call
New service-account key created in prod	High	Page on-call
Audit logging configuration changed	Critical	Page on-call
Multiple failed sign-ins on admin account	High	Page on-call
Public S3 bucket / GCS bucket created	High	Page on-call

Trigger	Severity	Destination
Secret-scanning alert (committed credential)	High	Page security
Container image with critical CVE deployed	High	Page platform

Add more for your stack. Each item should have:

- A configured alert rule (link / file path)
- A clear destination (specific channel / person / rotation)
- A documented runbook for the on-call to follow

How to gather the evidence

For each alert, screenshot or export:

- The rule definition in {{Datadog / CloudWatch / Sentinel / GCP Alerting}}
- The destination configuration in {{PagerDuty / Opsgenie}}
- A sample firing (anonymize sensitive data)

Save as `alert-<short-name>-<date>.{png,json}`.

Sample answer for the questionnaire

> Security-relevant events from cloud audit logs, IdP audit logs, and code-repository security alerts route to our on-call rotation via {{PagerDuty / Opsgenie}}. Alert rules and runbooks are listed in `alert-routing.md`. Triage during on-call follows the playbook in `../05-backups-recovery/incident-runbook.md`.

Refresh

- Alert config: annually unless you add new alerts.
- Sample fires: include any meaningful examples that show the chain works (don't manufacture them, but if real alerts fired, file an anonymized sample).

---

evidence-folder-template/02-logging-monitoring/audit-log-config.md

Audit log configuration

Evidence that audit logging is enabled across **every** production account / project / subscription — not just one. The most common auditor finding in this category is "logs missing from one of N accounts."

Evidence to keep here

1. **Config export** — the actual trail / sink configuration, in JSON or Terraform.
2. **Coverage proof** — a list of all production accounts / projects showing each is covered by the trail.
3. **Sample query result** — a query against the actual log storage returning records from each production account, proving logs are actually flowing (not just configured).

How to gather it

AWS

Org-level CloudTrail is the right answer:

```
# Confirm org-level trail exists and is logging
aws cloudtrail describe-trails --query 'trailList[?IsOrganizationTrail]'
aws cloudtrail get-trail-status --name <trail-name>

# Confirm every member account is covered
aws organizations list-accounts --query 'Accounts[?Status==`ACTIVE`].[Id,Name]' --output ta
```

Save the JSON output of `describe-trails` as `cloudtrail-org-config.json`.

GCP

Org-level audit log sink to BigQuery or GCS:

```
gcloud logging sinks list --organization=<org-id>
gcloud logging sinks describe <sink-name> --organization=<org-id>

# Verify Data Access logs are enabled where needed
gcloud projects get-iam-policy <project> --flatten='auditConfigs[]'
```

Save the sink config and IAM-policy audit-config as `gcp-audit-log-sink-config.json` and `gcp-data-access-logs.json`.

Azure

Diagnostic Settings or Activity Log Profile at the subscription / management-group level:

```
az monitor diagnostic-settings subscription list \
  --subscription <subscription-id>
```

Save the output as `azure-activity-log-config.json`.

Sample query proof

For each cloud, run a query that returns recent log events from each production account/project, and save a screenshot or CSV. This proves logs are flowing, not just that the config exists.

Example (CloudTrail / Athena):

```
SELECT accountid, COUNT(*) AS events, MIN(eventtime), MAX(eventtime)
FROM cloudtrail_logs
WHERE eventtime > date_add('day', -7, current_timestamp)
GROUP BY accountid;
```

Save the result as `cloudtrail-coverage-2026-MM-DD.csv`.

Example filenames

```
cloudtrail-org-config-2026-05-15.json
gcp-audit-log-sink-config-2026-05-15.json
azure-activity-log-config-2026-05-15.json
log-coverage-by-account-2026-05-15.csv
```

Refresh

Quarterly. Re-export the configs and re-run the coverage query. Drift (a new account that didn't get added to the trail) is the common failure mode.

---

evidence-folder-template/02-logging-monitoring/retention-policy.md

Audit log retention policy

Buyers and auditors want logs that cover the full review period (typically 12 months) AND are tamper-resistant. The evidence is the storage-lifecycle config + a query showing the oldest available record.

Evidence to keep here

1. **Lifecycle policy** — the rule on the storage bucket that retains audit logs for ≥ 365 days.
2. **Object-lock / immutability config** — proof that logs can't be tampered with even by an admin.
3. **Oldest-record query** — a query against the actual log storage showing the earliest available record date. This is the most-compelling evidence because it proves you didn't just set the policy yesterday.

How to gather it

AWS — S3 Object Lock + lifecycle

```
# Confirm object lock is enabled
aws s3api get-object-lock-configuration --bucket <log-bucket>

# Confirm lifecycle keeps records for retention period
aws s3api get-bucket-lifecycle-configuration --bucket <log-bucket>

# Oldest object in the bucket
aws s3api list-objects-v2 --bucket <log-bucket> \
  --query 'sort_by(Contents, &LastModified)[0]'
```

GCP — GCS Bucket Lock + retention

```
# Bucket retention policy + locked status
gsutil retention get gs://<log-bucket>

# Oldest object
gsutil ls -l gs://<log-bucket>/** | sort -k 2 | head -1
```

Azure — Storage immutability + retention

```
az storage account blob-service-properties show \
  --account-name <storage-account>
```

Sample answer for the questionnaire

> Production cloud audit logs are retained for **{{N}} days** in {{S3 / GCS / Storage Account}} with **object-lock / bucket-lock applied** to prevent tampering even by storage admins. The oldest available record dates from **{{date from your query}}**, covering the full review period.

Example filenames

```
s3-object-lock-config-2026-05-15.json
s3-lifecycle-policy-2026-05-15.json
oldest-record-query-2026-05-15.txt
```

Refresh

- Lifecycle / object-lock config: annually, or any time you change it.

- Oldest-record query: quarterly (it should keep moving forward as old logs age out per the lifecycle).

CI/CD & change management

evidence-folder-template/03-cicd-change-management/README.md

CI/CD and change management

Evidence that code changes to production go through review, approval, and leave an audit trail. SOC 2 CC8.1 territory; customer security questionnaires ask for this routinely.

What goes here

- Branch protection configuration (the rule that gates the production branch)
- Deploy approval configuration (GitHub Environments or equivalent)
- Release evidence — last 30 production deploys with author and approver
- Rollback procedure documentation

Owner

Engineering leads typically own the policy; **Platform** owns the CI/CD plumbing.

Common gotchas

- **Required reviews can be bypassed by admins.** Set "Do not allow bypassing the above settings" on GitHub branch protection — admins included.
- **Stale CODEOWNERS file.** Reviewers listed who left the company. Audit quarterly.
- **Production deploys from forks.** Without explicit environment protection, a fork PR can run production-context workflows. Use GitHub Environments + restrict secrets to the production environment.
- **Approval by author.** "Required PR review" should disallow approving your own PR. Confirm in the rule.

Cross-references

- Controls map: rows **3.1 – 3.5** in `../../controls-map.md`
- Platform guide for GitHub: `../../platforms/github.md`
- Questionnaire answers: questions 6-7 in `../../questionnaire-answer-examples.md`

---

evidence-folder-template/03-cicd-change-management/branch-protection.md

Branch protection

The rule that prevents direct pushes to the production branch and requires reviewed PRs. Evidence is the **actual rule definition** plus the CODEOWNERS file that determines who reviews what.

Evidence to keep here

1. **Rule export** — JSON of the branch protection or ruleset configuration on the production branch.
2. **CODEOWNERS file** — copy of the canonical CODEOWNERS at the time of review.
3. **Verification screenshot** — Settings → Rules / Branch Protection showing the rule is active and admins can't bypass.

How to gather it

GitHub — via API

```
# Branch protection (legacy)
gh api repos/<owner>/<repo>/branches/main/protection > branch-protection-main.json

# OR Rulesets (newer, more flexible)
gh api repos/<owner>/<repo>/rulesets > rulesets.json
gh api repos/<owner>/<repo>/rulesets/<ruleset-id> > ruleset-prod.json
```

Save the JSON. Also save CODEOWNERS:

```
cp <repo>/.github/CODEOWNERS codeowners-2026-05-15.txt
```

GitLab

```
glab api projects/<id>/protected_branches > protected-branches.json
```

Required rule contents

For SOC 2-friendly evidence, the rule should enforce:

- ☐ Pull request required for changes to `main` (no direct push)
- ☐ At least one review from a CODEOWNER (or designated team)
- ☐ Required status checks before merge (CI, tests, security scans)
- ☐ Cannot approve your own PR
- ☐ Linear history (optional but auditor-friendly)
- ☐ Required signed commits (optional, but a credibility booster)
- ☐ Cannot be bypassed by admins (GitHub: "Do not allow bypassing")

If any of these is off, document why with compensating control or accept it as a known gap and target a fix date.

Sample answer for the questionnaire

> The production branch `main` is protected by a GitHub Ruleset (see `ruleset-prod-<date>.json`) requiring:
>
> - PR review from a CODEOWNER
> - Passing CI status checks (lint, type-check, tests, security scans)
> - No bypass — admins included

> CODEOWNERS file is at `<repo>/.github/CODEOWNERS` and reviewed quarterly during the IAM review.

Example filenames

```
branch-protection-main-2026-05-15.json
ruleset-prod-2026-05-15.json
codeowners-2026-05-15.txt
branch-protection-screenshot-2026-05-15.png
```

Refresh

Quarterly. Re-export the rule; verify CODEOWNERS doesn't reference former employees.

---

evidence-folder-template/03-cicd-change-management/deploy-approvals.md

Deploy approvals

Branch protection gates the merge; deploy approvals gate the actual production push. SOC 2 wants both: code review (3.1) AND deployment approval (3.2). On GitHub, that means a protected Environment with required reviewers.

Evidence to keep here

1. **Environment config** — JSON or screenshot of the production environment's protection rules.
2. **Approver list** — who can approve a deploy to production. This is a sensitive list; review quarterly.
3. **Sample approval trail** — 3-5 recent production deploys showing the approver and approval timestamp.

How to gather it

GitHub Environments

```
gh api repos/<owner>/<repo>/environments/production > environment-prod.json
gh api repos/<owner>/<repo>/environments/production/deployment-protection-rules \
  > protection-rules-prod.json
```

Required reviewers list:

```
gh api repos/<owner>/<repo>/environments/production \
  --jq '.protection_rules[] | select(.type=="required_reviewers")'
```

GitLab — Protected Environments

```
glab api projects/<id>/protected_environments > protected-envs.json
```

Deploy approval trail

Pull the last 30 production deploys with author + approver:

```
gh run list --workflow=deploy.yml --branch=main --limit=30 \
  --json databaseId,displayTitle,event,createdAt,actor,conclusion \
  > recent-deploys.json
```

For each, the deploy review record is at:

```
gh api repos/<owner>/<repo>/actions/runs/<run-id>/approvals
```

Aggregate into a CSV or table for `release-evidence.md`.

Required protection contents

- [ ] At least one required reviewer on `production` environment
- [ ] Reviewer can't be the PR/commit author (or document why if allowed)
- [ ] Production secrets scoped to the `production` environment only (not repository-wide)
- [ ] Deploy branch / tag restriction (only `main` or release tags)

Sample answer for the questionnaire

> Production deploys are gated by a GitHub Environment named `production` with required reviewers. The approver list is in `deploy-approvals.md` and rotates

> with team changes. Production secrets are scoped to that environment only — they are not accessible from arbitrary workflows.

Example filenames

```
environment-prod-2026-05-15.json
protection-rules-prod-2026-05-15.json
recent-deploys-2026-05-15.json
```

Refresh

Quarterly. Re-export the environment config; verify the reviewer list against the IAM quarterly review.

---

evidence-folder-template/03-cicd-change-management/release-evidence.md

Release evidence

The audit trail of "who shipped what to production, when, and who approved it." This is the most-requested piece of change-management evidence on questionnaires.

Evidence to keep here

A quarterly export of the last 30+ production deploys with:

- Date / time
- Commit SHA + PR link
- Author
- Approving reviewer (from branch protection)
- Approving deployer (from environment protection)
- Deploy outcome (success / rolled back)

How to gather it

GitHub — combined query

```
gh run list \
  --workflow=deploy.yml \
  --branch=main \
  --limit=50 \
  --json databaseId,displayTitle,headSha,headBranch,createdAt,actor,conclusion,event \
  > recent-runs.json
```

For each run, get the PR + reviewers:

```
gh api repos/<owner>/<repo>/commits/<sha>/pulls --jq '.[0]' > pr.json
gh api repos/<owner>/<repo>/pulls/<pr-number>/reviews \
```

```
  --jq '[.[] | select(.state=="APPROVED")]' > approvers.json
```

Combine into a single CSV / table:

Date	SHA	PR	Author	PR approver	Deploy approver	Outcome
2026-05-15	a1b2c3	#1234	alice	bob	carol	success

Save as `release-evidence-2026-Q2.csv`.

Sample answer for the questionnaire

> Every production deploy is associated with a Pull Request, a CODEOWNER review, and a deploy-approver. The most recent 30 production deploys are exported quarterly to `release-evidence-<quarter>.csv`. The full history is reconstructable from `git log` on the `main` branch combined with the CI/CD platform's deploy history.

Bonus: signed commits

If your CODEOWNERS sign their commits and your branch rule requires signature verification, mention that — it's a credibility booster and not a heavy lift to adopt.

```
# Verify
gh api repos/<owner>/<repo>/commits/<sha> --jq '.commit.verification'
```

Example filenames

```
release-evidence-2026-Q2.csv
recent-runs-2026-05-15.json
```

Refresh

Quarterly. Re-run the export against the last 90 days.

Vulnerability & secrets

evidence-folder-template/04-vulnerability-secrets/README.md

Vulnerability and secrets hygiene

Evidence that you scan for known vulnerabilities and exposed secrets — and that you actually do something about the findings. "Scanner enabled" is not enough; auditors want the remediation record.

What goes here

- Dependency scanning configuration + remediation log
- Container / image scanning configuration + recent scan results
- Secret scanning configuration + remediation history
- Pen-test / red-team report (if any)

Owner

Platform typically owns scanner configuration; **Security** owns triage and remediation policy. **Engineering** owns the actual fixes.

Common gotchas

- **Scanner enabled, alerts ignored.** A flood of unread Dependabot alerts is worse than no scanner because it documents that you knew and didn't act. Set an SLA and triage actively.
- **No bypass policy.** When a critical CVE blocks a deploy, what's the documented exception process? "We just merged it" is not an acceptable answer; "Documented exception in the PR with mitigation
  - remediation timeline" is.
- **Image scanning only at build, not deploy.** A long-lived image may pass scan at build then accumulate CVEs as new ones are disclosed. Periodic rescan of in-production images closes this.

- **Secret-scanning without push protection.** Detection after a commit happens is fine; **prevention** of the push is better. GitHub Advanced Security supports push protection.

Cross-references

- Controls map: rows **4.1 – 4.4** in `../../controls-map.md`
- Platform guides: per-platform scanner details in `../../platforms/`
- Questionnaire answers: questions 8-9 in `../../questionnaire-answer-examples.md`

---

evidence-folder-template/04-vulnerability-secrets/dependency-scanning.md

Dependency scanning

Evidence that dependencies are scanned for known CVEs continuously, not once-and-forgotten — and that there's a written SLA for remediation.

Evidence to keep here

1. **Scanner configuration** — Dependabot / Renovate / Snyk config file or repo settings.
2. **Remediation SLA** — written policy on how fast each severity gets fixed.
3. **Recent remediation log** — last quarter's CVE alerts with their resolution (fixed via PR / accepted with mitigation / false positive).

Recommended SLA

A defensible SLA, in plain language:

Severity	SLA
Critical (CVSS ≥ 9.0, exploited in the wild)	Patched within **7 days**
High (CVSS 7.0-8.9)	Patched within **30 days**
Medium	Patched within **90 days** or accepted with documented mitigation
Low	Patched at convenience; reviewed quarterly

Match the wording to your actual practice. If you can't hit 7 days for critical, document the real SLA and the compensating control (WAF rule, feature flag off, etc.).

How to gather it

GitHub Dependabot

```
# Config in repo
cp <repo>/.github/dependabot.yml dependabot-config-2026-05-15.yml

# Open + closed alerts
gh api repos/<owner>/<repo>/dependabot/alerts \
  --jq '[.[] | {number, severity, state, package: .dependency.package.name, fix_resolved: .f
  > dependabot-alerts-2026-05-15.json
```

Renovate

The Renovate config in `renovate.json` or `.github/renovate.json`. The dashboard issue (if you have one) is the log of merges and deferrals.

Snyk

```
snyk monitor --all-projects
# Then export from the Snyk web UI: Reports → Issues
```

Remediation log format

For each quarter:

```
# Dependency remediation — 2026 Q2

| Date       | CVE           | Severity | Package     | Action
| ---------- | ------------- | -------- | ----------- | ----------------------------------
| 2026-04-12 | CVE-2026-1234 | Critical | lodash      | Patched in #1234
| 2026-04-15 | CVE-2026-2345 | High     | next        | Patched in #1240
```

```
| 2026-05-02 | CVE-2026-3456 | Medium   | tailwindcss | False positive (not used)
| 2026-05-10 | CVE-2026-4567 | Medium   | undici      | Mitigation in place; tracked for n
```

Sample answer for the questionnaire

> Dependencies are scanned continuously by {{Dependabot / Renovate / Snyk}}. Critical CVEs are remediated within 7 days, high within 30 days, medium within 90 days. See `dependency-scanning.md` for the SLA and the most recent remediation log.

Example filenames

```
dependabot-config-2026-05-15.yml
dependabot-alerts-2026-05-15.json
remediation-log-2026-Q2.md
```

Refresh

Quarterly remediation log. SLA changes only when policy changes.

---

evidence-folder-template/04-vulnerability-secrets/image-scanning.md

Container / image scanning

Evidence that container images destined for production are scanned for CVEs before they ship, and that the deploy blocks on critical findings.

Evidence to keep here

1. **Scanner config** — Trivy / Grype config in CI, or Artifact Registry / ECR scan settings.
2. **CI integration** — the pipeline step that runs the scan and gates the deploy.
3. **Recent scan results** — last 30 deploys' scan outcomes.
4. **Exception process** — documented procedure for shipping despite a critical CVE (must be rare and require approval).

How to gather it

Trivy (open source, CI-friendly)

CI step example:

```
- name: Scan image with Trivy
  uses: aquasecurity/trivy-action@<pinned-sha>
  with:
    image-ref: ghcr.io/${{ github.repository }}:${{ github.sha }}
    format: sarif
    output: trivy-results.sarif
    severity: CRITICAL,HIGH
    exit-code: 1 # Fail the build on findings
```

Save a sample scan output (sanitized) as `trivy-sample-output-2026-05-15.sarif`.

AWS — ECR scan

```
aws ecr describe-image-scan-findings \
  --repository-name <repo> \
  --image-id imageDigest=<digest> > ecr-scan-result.json
```

Enable scan-on-push at the repository level:

```
aws ecr put-image-scanning-configuration \
  --repository-name <repo> \
  --image-scanning-configuration scanOnPush=true
```

GCP — Artifact Registry / Container Analysis

```
# Enable vulnerability scanning (one-time)
gcloud services enable containeranalysis.googleapis.com

# Pull scan results for a specific image
gcloud artifacts docker images describe <image-uri> \
  --show-package-vulnerability
```

Exception process

Critical CVEs sometimes can't be patched immediately (upstream not released, breaking change required). The defensible answer:

1. PR opened acknowledging the CVE.
2. Mitigation in place — WAF rule, feature flag off, network isolation, etc.
3. Tracked exception with a remediation date.
4. Approver other than the PR author signs off.

File the exception in this folder as `cve-exception-<cve>-<YYYY-MM-DD>.md`.

Sample answer for the questionnaire

> Container images for production are scanned in CI by {{Trivy / ECR / Artifact Registry}}. Builds fail on critical or high CVEs unless an explicit, approved exception is filed (process in `image-scanning.md`). Last 30 deploys' scan outcomes are exported quarterly to `recent-scans-<quarter>.csv`.

Example filenames

```
trivy-config-2026-05-15.yml
trivy-sample-output-2026-05-15.sarif
recent-scans-2026-Q2.csv
cve-exceptions-2026-Q2.md
```

Refresh

Quarterly export of recent scans. Re-confirm scanner is gating deploys at the CI level (not just running advisory).

---

evidence-folder-template/04-vulnerability-secrets/secret-scanning.md

Secret scanning

Evidence that credentials accidentally committed to repos are detected — ideally **blocked at push time** — and triaged within a documented SLA.

Evidence to keep here

1. **Scanner config** — GitHub Advanced Security secret scanning + push protection settings, or equivalent.
2. **Remediation SLA** — written policy on triage time + key rotation.
3. **Remediation log** — sample of recent secret-scanning hits with their resolution.

Recommended SLA

When a secret is detected:

1. **Within 1 hour**: revoke the secret at the source (don't wait for the commit to be rewritten).
2. **Within 4 hours**: rotate any related credentials that may have been derived.
3. **Within 24 hours**: rewrite git history if the secret was pushed publicly (BFG / git-filter-repo) and confirm scrub.
4. **Within 1 week**: postmortem if the secret reached production credentials.

The "revoke first, then clean up" order matters: the moment a secret is committed publicly, it's compromised. Git history rewrite is optional cleanup, not defense.

How to gather it

GitHub Advanced Security

```
# Check the org / repo has secret scanning + push protection enabled
gh api repos/<owner>/<repo> --jq '.security_and_analysis'

# List alerts
gh api repos/<owner>/<repo>/secret-scanning/alerts \
  --jq '[.[] | {number, secret_type, state, resolution, created_at, resolved_at}]' \
  > secret-scanning-alerts-2026-05-15.json
```

GitLab

```
glab api projects/<id>/secrets > secrets-config.json
```

Standalone — trufflehog / gitleaks in CI

If you don't have GitHub Advanced Security, run a scanner in CI:

```
- name: Secret scan
  uses: gitleaks/gitleaks-action@<pinned-sha>
```

Remediation log format

```
# Secret-scanning remediation — 2026 Q2

| Date       | Type           | Source          | Action                                |
| ---------- | -------------- | --------------- | ------------------------------------- |
| 2026-04-08 | AWS access key | feature branch  | Revoked + rotated; not pushed to main  |
| 2026-05-22 | OpenAI API key | example in docs | Replaced with placeholder; key revoked |
```

Sample answer for the questionnaire

> GitHub Advanced Security secret scanning is enabled across the org with push protection — pushes containing detected secrets are blocked. Triage is per the SLA in `secret-scanning.md` (revoke in 1 hour; rotate dependents in 4 hours). Recent remediation log attached as `remediation-log-<quarter>.md`.

Example filenames

```
secret-scanning-config-2026-05-15.json
secret-scanning-alerts-2026-05-15.json
remediation-log-2026-Q2.md
```

Refresh

Quarterly remediation log + verification that push protection is still on (it can be toggled off accidentally).

Backups & recovery

evidence-folder-template/05-backups-recovery/README.md

Backups, recovery, and availability

Evidence that data can be recovered — verified by actual restore tests, not just backup-job success messages. SOC 2 CC9.1 and A1.x; customer security reviews ask the "have you tested restore?" question routinely.

What goes here

- Backup configuration (managed database backups, retention)
- Restore-test runbook + most recent test report
- Incident runbook
- Status page / availability signal

Owner

Platform / SRE owns backup configuration and restore tests. **Engineering leads** own the incident runbook.

Common gotchas

- **Backups confirmed but never restored.** "Backup succeeded last night" doesn't prove the backup is usable. Auditors want a restore test — actual data restored to a non-production environment, with row counts and runtime documented.
- **RPO / RTO defined in slides, not in code.** The recovery point + recovery time objectives need to be measurable against the actual restore test outcome.
- **No incident runbook for the major-incident case.** Every team has runbooks for routine stuff; few have the "production database is corrupt" version. Write the bad-day runbook before you need it.

Cross-references

- Controls map: rows **5.1 – 5.4** in `../../controls-map.md`
- Questionnaire answers: questions 10-11 in `../../questionnaire-answer-examples.md`

---

evidence-folder-template/05-backups-recovery/backup-config.md

Backup configuration

The configured backup schedule and retention for production data stores. Evidence is the actual managed-service configuration plus a recent record of successful backups.

Evidence to keep here

1. **Configuration export** — for each production data store (databases, object storage, message queues with state).
2. **Last 7 days of backup success records** — proves it's actually running, not just configured.
3. **Retention policy** — how long backups are kept and where they live.

How to gather it

AWS — RDS

```
aws rds describe-db-instances --db-instance-identifier <id> \
  --query 'DBInstances[0].{BackupRetentionPeriod:BackupRetentionPeriod, PreferredBackupWin

aws rds describe-db-snapshots --db-instance-identifier <id> \
  --snapshot-type automated --max-records 10
```

Save as `rds-backup-config-<id>-2026-05-15.json`.

GCP — Cloud SQL

```
gcloud sql instances describe <instance> \
  --format='value(settings.backupConfiguration)'
```

```
# Backup history
gcloud sql backups list --instance=<instance> --limit=10
```

Azure — SQL / Database

```
az sql db show --resource-group <rg> --server <server> --name <db> \
  --query 'currentBackupStorageRedundancy'

az sql db ltr-backup list --resource-group <rg> --server <server> --database <db>
```

Retention guidance

For SOC 2-friendly evidence, typical answers:

- **Operational backups**: 7-35 days of point-in-time recovery (managed by your cloud database).
- **Long-term retention**: monthly or quarterly snapshots retained for 1+ year, stored in a separate region or account.
- **Encryption**: at rest (default for managed services) + at least cross-account/cross-project access controls so a compromised primary account doesn't lose backups too.

Sample answer for the questionnaire

> Production databases are backed up automatically by {{RDS / Cloud SQL / Azure SQL}}. Point-in-time recovery is enabled with **{{N}}-day** retention; long-term snapshots are retained for **{{N}} year**. Backups are encrypted at rest and stored {{in a separate region / account}}. Backup success is monitored daily; restore is tested quarterly (see `restore-test.md` for the most recent test result).

Example filenames

```
rds-backup-config-prod-db-2026-05-15.json
cloudsql-backup-history-2026-05-15.json
backup-success-log-2026-05-15.csv
```

Refresh

Configuration: annually unless changed. Success log: quarterly spot-check.

---

evidence-folder-template/05-backups-recovery/incident-runbook.md

Incident runbook

The "what to do when something is on fire in production" document. Auditors and customer security reviews ask for it; the bigger value is that writing it forces the team to think through worst cases before they happen.

Evidence to keep here

1. **This runbook** — kept current.
2. **Recent postmortems** (if any) — sanitized for sharing externally if needed.
3. **On-call rotation** — current schedule + escalation path.

Master runbook

```
# Incident Runbook

## Severity definitions

| Sev  | Definition                                                          | Response time
| ---- | ------------------------------------------------------------------- | -------------
| SEV1 | Customer-facing service down, major data loss, or security breach | Page immediat
| SEV2 | Significant degradation for many customers; potential data issue  | Page on-call; 
| SEV3 | Limited customer impact OR internal-only issue                     | Ticket; addre

## On-call rotation

- Current schedule: {{PagerDuty rotation link}}
- Primary on-call: see PagerDuty
- Secondary / escalation: see PagerDuty
- Engineering lead escalation: {{name}}
- Security escalation: {{name}}

## SEV1 response procedure

1. **Acknowledge the page** within 5 minutes.
2. **Declare an incident** — post in {{#incidents channel}} with one-line description + se
```

```
3. **Spin up a war room** — {{Slack huddle / Zoom / dedicated
   incident channel}}. Bring in: on-call, eng lead, anyone with relevant context.
4. **Assign roles**:
   - **Incident commander** — coordinates, decides
   - **Operator** — drives the actual fix
   - **Comms** — customer notifications, status page, internal updates
5. **Triage**:
   - What's broken? (specific service / API / data store)
   - Blast radius? (customer count, data category)
   - Is data being lost or exposed? (security implication?)
6. **Stabilize first, root-cause later**:
   - Roll back to last known good if possible
   - Failover to standby region if applicable
   - Enable read-only mode if write-side is corrupt
7. **Notify** customers via {{StatusPage / email / in-app}} per the threshold:
   - SEV1: notify within 30 min
   - SEV2: notify within 2 hours
8. **Resolve** and confirm: monitoring back to baseline; spot-check the affected flows
9. **Postmortem** — schedule within 5 business days; sanitize and file in this folder

## Special-case runbooks

### Production database corruption / data loss

1. Take application offline (read-only mode or 503 page)
2. Identify last known good snapshot
3. Follow `restore-test.md` procedure with production target
4. Validate restored DB
5. Bring application back up
6. Communicate data-loss window to affected customers

### Security incident (suspected breach / credential exposure)

1. Revoke the suspect credential immediately (don't wait to investigate further)
2. Rotate all related credentials (derive risk: what could have been accessed?)
3. Pull cloud audit logs for the affected time window
4. Engage security escalation: {{name}}
5. If customer data potentially exposed: notify legal + privacy/comms within 24 hours
6. Document everything; do not delete evidence

### Major cloud-provider outage

1. Confirm the outage on the provider's status page (rule out our own bug)
2. Switch traffic to standby region if multi-region
```

````
3. Notify customers via status page
4. Wait + retry plan documented for stateful services

## Postmortem template

```markdown
# Incident Postmortem — YYYY-MM-DD

**Severity:** SEV1 / SEV2 / SEV3 **Duration:** {{start time}} → {{end time}} ({{X minutes}}
**Customer impact:** {{description}}

## Timeline

| Time (UTC) | Event                 |
| ---------- | --------------------- |
| HH:MM      | Alert fired           |
| HH:MM      | On-call acknowledged  |
| HH:MM      | War room opened       |
| HH:MM      | Root cause identified |
| HH:MM      | Mitigation applied    |
| HH:MM      | Confirmed recovered   |

## Root cause

{{What broke and why}}

## What worked

{{Things the team did well}}

## What didn't

{{Things to improve}}

## Action items

| Item | Owner | Due |
| ---- | ----- | --- |
| ...  | ...   | ... |
```
````

Refresh

Annually unless a real incident happens (in which case the postmortem informs runbook updates).

---

evidence-folder-template/05-backups-recovery/restore-test.md

Restore test

The single piece of evidence that most teams don't have but most buyers ask for: **a documented restore from backup**, recently, with measured outcomes. "We back up nightly" is not enough; an auditor wants to see the proof you can restore.

Evidence to keep here

For each quarterly test:

1. **Restore-test runbook** — the procedure followed.
2. **Test report** — date, source backup, target environment, row counts, integrity-check outcome, wall-clock time.
3. **Sign-off** — name of the person who validated the restore.

Restore-test runbook (master template)

```
# Restore Test Runbook — Production Database

## Pre-conditions

- Non-production environment available with capacity for full DB
- Test runner has read access to backups + write access to target
- Notifications: post in {{#engineering channel}} that a test is starting

## Steps

1. Choose source backup: most recent automated snapshot for prod DB
2. Target: dedicated `restore-test` environment in same cloud region
3. Initiate restore:
   - {{cloud-specific command}}
4. Wait for restore to complete; record wall-clock duration
5. Validate:
   - Row counts on key tables match expected within 5% (writes during snapshot interval ca
     small skew)
```

```
   - Integrity checks: foreign-key constraint validation, application boot success against
     restored DB
   - Spot-check a recent record (e.g., user signup from yesterday)
6. Tear down: delete the restored DB instance (cost control)
7. File the report

## Acceptance

- Restore completes in ≤ RTO ({{X hours}})
- Application boots against restored DB
- No data integrity errors
```

Test report (per-test template)

```
# Restore Test — 2026-05-15

**Tester:** {{name}}, **Reviewer:** {{name}}

| Item                | Value                                                 |
| ------------------- | ----------------------------------------------------- |
| Source backup       | Automated snapshot 2026-05-14T03:00Z                  |
| Source DB           | prod-main (RDS / Cloud SQL)                           |
| Target environment  | restore-test (same region, same VPC)                  |
| Restore initiated   | 2026-05-15T14:00Z                                     |
| Restore completed   | 2026-05-15T14:38Z                                     |
| **Wall-clock time** | **38 min**                                            |
| Row count: users    | 142,318 (expected ~142,300; delta within 0.05%)       |
| Row count: orders   | 1,084,221 (expected ~1,084,000; delta within 0.02%)   |
| FK integrity        | PASS                                                  |
| Application boot    | PASS — read-only smoke test against restored DB       |
| Tear-down           | Completed 2026-05-15T15:10Z                           |

## Outcome

PASS. Restore time 38 min, well within RTO (4 hours). No integrity issues.

## Issues / follow-ups

{{None / Issue list}}
```

RPO / RTO answer

Once you have a real restore test:

> **RPO**: {{N}} minutes — point-in-time recovery is enabled, so the theoretical worst-case data loss is one continuous-backup interval.
>
> **RTO**: {{N}} hours — validated by the quarterly restore test (most recent: {{date}}, {{minutes}} wall-clock).

Refresh

Quarterly is the standard for SOC 2 readiness. The test takes 1-2 hours; setting a recurring calendar event is the cheapest durable improvement most teams can make.

AI workload controls

evidence-folder-template/06-ai-workload-controls/README.md

AI workload controls

The AI-specific controls customer procurement teams have started asking about: who can call the model endpoints, what gets logged, how data is partitioned, how spend is capped, and what happens when something goes wrong.

If your product doesn't ship AI features, skip this folder.

What goes here

- Endpoint authentication evidence
- Prompt / tool-call logging policy
- Per-tenant spend caps + abuse runbook
- Data retention + tenant isolation for retrieval / embeddings

Owner

AI team (or platform team where AI work lives) for technical controls; **Security** for policy; **Legal** for retention and DPA alignment.

Common gotchas

- **Public model endpoints with no auth.** Even in pre-launch, a public inference URL without rate-limiting or auth is a spend and abuse risk. Auditors increasingly catch this.
- **Logging that captures full prompts and outputs without a retention policy.** Inference content can contain customer data — keep it, but bound it.
- **Tenant data crossing in RAG.** Multi-tenant vector indices without namespace partitioning leak embeddings across tenants. Either index per-tenant or filter at query time and prove the filter is enforced.

- **No spend cap on the underlying model.** A misbehaving feature or compromised key can burn through monthly budget overnight. Per-tenant quota at the gateway is the answer.
- **No abuse playbook.** What do you do when a prompt-injection attack happens? When a customer reports their data appeared in another customer's response? Write the playbook before you need it.

Mapping to standards

NIST AI RMF and ISO/IEC 42001 are the emerging references for AI governance. The controls here map roughly to NIST AI RMF **MANAGE** and **MEASURE** functions.

Cross-references

- Controls map: rows **6.1 – 6.6** in `../../controls-map.md`
- Questionnaire answer: question 12 in `../../questionnaire-answer-examples.md`

---

evidence-folder-template/06-ai-workload-controls/data-retention.md

AI data retention and tenant isolation

Evidence that retrieval indices, embeddings, prompt logs, and other AI-derived data are partitioned per-tenant and retained per customer-DPA terms.

Evidence to keep here

1. **Tenant-isolation policy** — how each tenant's data is partitioned in the AI stack.
2. **Retention policy** — what's kept, for how long, and where.
3. **DPA alignment doc** — mapping of contract terms to actual retention windows.

Tenant-isolation policy

For each AI-derived data store, document:

Vector / retrieval index

Tenant model	How
Index-per-tenant	Separate Pinecone index / Weaviate class / pgvector schema per tenant. Strongest isolation.
Namespace-per-tenant	Single index with tenant-scoped namespaces / metadata filters. Simpler ops; weaker isolation; the query must enforce the namespace at the application layer.
Multi-tenant with filter	Single index, filter at query. Lowest isolation; requires bulletproof filter logic + regular testing.

Evidence: the actual configuration plus a test that proves cross-tenant queries return zero results.

Embedding cache

If you cache embeddings to reduce model spend, keep per-tenant caches. A cross-tenant cache can leak content shape.

Conversation / session history

Stored where? With what tenant scoping? Retention window?

Retention policy

Data	Retention	Justification
Inference request logs (metadata)	365 days	Operational + audit
Sampled inference content	30 days	Quality + safety review
Flagged-for-review content	90 days	Active incident investigation
Vector / RAG index	Indefinite (tied to product use)	Functional
Conversation history	Per customer setting (default 30 days)	Product UX + privacy

DPA alignment

For each enterprise customer with a custom DPA, file a one-page crosswalk:

```
# DPA crosswalk — {{Customer name}} — signed 2026-MM-DD

| DPA term                    | Our default      | Honored by
| --------------------------- | ---------------- | ----------------------------------------
```

```
| Logs retention ≤ 30 days     | 365 days default | Shortened to 30d for this tenant via pe
| No training on customer data | Same             | We don't train on customer data org-wi
| Data residency: EU           | Multi-region     | Routed to EU-only region for this tena
| Right to delete: 7-day SLA   | N/A default      | Manual process documented in `delete-r
```

How to gather the evidence

- Vector store config: export from Pinecone / Weaviate / pgvector showing the partitioning model.
- Negative test: pick two tenants A and B; query A's namespace with B's tenant ID and verify zero results.
- DPA crosswalks: one per enterprise customer with custom terms.

Sample answer for the questionnaire

> Per-tenant data in our AI stack is partitioned via {{model: namespace / separate index / filter}}. Inference request metadata is retained 365 days; sampled content is retained 30 days; flagged-for-review content is retained 90 days with security-team-only access. We do **not** use customer data to train or fine-tune base models. Retention is aligned to customer DPA terms — stricter customer terms override our defaults. See `data-retention.md`.

Example filenames

```
vector-store-tenant-isolation-2026-05-15.md
ai-retention-policy-2026.md
dpa-crosswalk-customer-acme-2026-03-15.md
```

Refresh

Policy: annually. Tenant-isolation test: quarterly. DPA crosswalks: per signed contract.

evidence-folder-template/06-ai-workload-controls/endpoint-auth.md

Model endpoint authentication

Evidence that every inference endpoint requires authentication, that unauthenticated requests are denied, and that the auth mechanism ties requests back to a tenant for downstream controls (logging, spend caps, abuse handling).

Evidence to keep here

1. **Endpoint inventory** — every model-serving URL / route.
2. **Auth configuration** — gateway / API key / OAuth / IAM config per endpoint.
3. **Denial proof** — sample request without auth showing a 401/403.
4. **Tenant attribution** — how each authenticated request maps to a tenant ID.

How to gather it

If using a cloud provider's hosted model (OpenAI, Vertex AI, Bedrock)

The customer-facing endpoint is *your* application API, not the provider's API. The application API does the auth; the provider's API gets called server-side with your single org key.

Evidence:

```
# Sample denied request to your inference route
curl -i https://api.example.com/v1/generate -d '{"prompt": "test"}'
# → 401 Unauthorized
```

Plus the auth middleware code or config:

```
# In your API gateway or framework, the auth requirement
```

```
# Save as middleware-config.md
```

If self-hosting (Triton, vLLM, Ollama)

The model server must NOT be publicly reachable. Run it behind:

- Private VPC subnet only
- An API gateway that authenticates the caller
- IP allowlist for known traffic origins

Evidence: the network configuration (security group / firewall rules) + the auth gateway config.

Tenant attribution

Each authenticated request should carry a tenant identifier into logging and spend tracking. Example log line shape:

```
{
  "ts": "2026-05-15T14:00:00Z",
  "tenant_id": "tenant_abc123",
  "user_id": "user_456",
  "endpoint": "/v1/generate",
  "model": "gpt-4o-mini",
  "input_tokens": 1234,
  "output_tokens": 567,
  "request_id": "req_xyz"
}
```

The `tenant_id` is what later controls (spend caps, abuse runbook, data retention) key off.

Sample answer for the questionnaire

> All model inference endpoints require authentication via {{API key / OAuth / IAM}} at our application gateway. Unauthenticated requests receive HTTP 401. Each authenticated request is tagged with a tenant identifier that propagates into logging and quota systems. The provider model endpoints (OpenAI / Vertex AI / Bedrock) are called server-side using infrastructure credentials only — they are never exposed to end users.

Example filenames

```
endpoint-inventory-2026-05-15.md
gateway-auth-config-2026-05-15.json
denied-request-sample-2026-05-15.txt
```

Refresh

Quarterly. New endpoints get added; old ones can drift. Audit the inventory.

---

evidence-folder-template/06-ai-workload-controls/prompt-tool-call-logging.md

Prompt / tool-call logging

Evidence that prompts, tool calls, and outputs are logged with a policy that bounds retention and access — so investigations are possible without becoming a long-term data risk.

Evidence to keep here

1. **Logging policy** — what's logged, retention, who can read it.
2. **Sample log entry** — sanitized.
3. **Access controls** — who can query the logs.

Logging policy template

```
# AI Inference Logging Policy

## What we log

For every inference request:

- Timestamp, request ID, tenant ID, user ID
- Endpoint, model, input/output token counts
- Latency, status code

For inference requests **flagged for review** (per the abuse runbook):

- Full prompt
- Full output (or hash, depending on customer DPA)
- Tool calls made and their results

For routine traffic:

- {{policy: store hashes of prompt + output, retain full content for N days, etc.}}

## What we don't log
```

```
- {{e.g. customer PII unless required for incident investigation}}
- {{e.g. authentication tokens, API keys}}

## Retention

| Log type                      | Retention                                  |
| ----------------------------- | ------------------------------------------ |
| Request metadata (no content) | 365 days                                   |
| Sampled prompt+output content | 30 days                                    |
| Flagged-for-review content    | 90 days, with security-team access only |

## Access

- Application engineers: metadata logs only
- AI / safety team: metadata + sampled content
- Security: all logs, for active investigations

Reading the flagged-content store requires an access-review entry (who, when, why).

## Customer data alignment

This policy must align with our customer DPA terms. If a customer contract states "no AI t
customer data" or "30-day retention," our policy must respect the stricter term.
```

How to gather sample evidence

```
# Pull a sanitized sample log entry from your logging backend
# (Datadog / Honeycomb / Loki / native)
# Replace any actual content with [REDACTED] before saving
```

Sample answer for the questionnaire

> Inference requests are logged centrally with retention bounded per the policy in `prompt-tool-call-logging.md`. Metadata (timestamps, token counts, tenant IDs) is retained for 365 days; content is sampled for 30 days; flagged-for-review

> content is retained for 90 days with security-team-only access. Retention windows align with customer DPA terms; we do not retain customer prompts longer than contracted.

Example filenames

```
inference-logging-policy-2026.md
sample-log-entry-sanitized-2026-05-15.json
log-access-roles-2026-05-15.md
```

Refresh

Policy: annually unless changed. Sample log: quarterly.

---

evidence-folder-template/06-ai-workload-controls/spend-caps.md

Spend caps and abuse handling

Evidence that AI features have per-tenant rate-limits and spend caps so a single tenant (or a misbehaving feature) can't burn through monthly model budget, and that there's a documented runbook for abuse and incidents.

Evidence to keep here

1. **Spend-cap configuration** — per-tenant or per-org quotas at the gateway.
2. **Alert routing** — what fires when caps are hit.
3. **Abuse runbook** — what to do when a customer reports prompt-injection, data leakage, or hostile use.

Spend-cap configuration

The gateway in front of model endpoints should enforce:

Tier	Per-minute rate limit	Daily token cap	Monthly spend cap
Free tier	60 req/min	50k tokens	$0 (hard stop)
Paid tier	600 req/min	1M tokens	per contract
Enterprise	6000 req/min	10M tokens	per contract

Evidence to keep:

- Rate-limit / quota config (gateway / cloud-provider quotas)
- Logs showing limits actually firing for at least one tenant
- Alert rules that fire to on-call when:
  - A single tenant exceeds 80% of monthly cap

- Org-wide spend exceeds budget threshold
- Unusual spike (e.g., >5σ above tenant's baseline)

Provider-level guardrails (defense in depth)

Even with your gateway, set caps at the model-provider level too — in case the gateway is bypassed:

- **OpenAI**: Org → Usage limits → Hard / Soft caps per project
- **Vertex AI**: Quotas → API requests per minute, tokens per project
- **Bedrock**: Service quotas + CloudWatch billing alarms

Save the config exports.

Abuse runbook

```
# AI Abuse Response Runbook

## Triggers

1. **Spend anomaly** — alert fires for spike or cap-approach
2. **Customer report** — "your AI said something inappropriate" / "your AI leaked another c
   data" / "prompt injection attack succeeded"
3. **Internal review flag** — content moderation or safety review surfaces something

## Response (Sev1 — data leak or breach)

1. Acknowledge within 15 minutes
2. **Throttle or disable** the affected endpoint immediately if data leakage is suspected
   break the feature than leak more)
3. Pull recent inference logs for the affected tenant + the complaining customer
4. Engage security escalation + legal escalation
5. Customer notification per breach policy

## Response (Sev2 — misuse / abusive prompts)

1. Acknowledge within 1 hour
```

```
2. Identify the offending tenant / user
3. Apply rate-limit reduction or temporary block
4. Document with a per-incident write-up

## Response (Sev3 — spend anomaly only, no leakage)

1. Acknowledge within 4 hours
2. Identify the runaway feature or tenant
3. Apply quota reduction; investigate root cause (loop? compromised key?)
4. File a postmortem if the root cause is engineering-side
```

Sample answer for the questionnaire

> AI features run behind a gateway that enforces per-tenant rate limits and monthly spend caps. Quotas are documented in `spend-caps.md`. When caps are approached or anomalies are detected, alerts page on-call. Abuse and misuse responses follow the runbook in `spend-caps.md` (also in the abuse section).

Example filenames

```
gateway-quotas-2026-05-15.yaml
provider-spend-limits-2026-05-15.json
abuse-runbook-2026.md
incident-log-2026-Q2.md
```

Refresh

Quarterly. Confirm caps still match contract tiers; review any incident reports.

Platforms

platforms/aws.md

AWS evidence guide

Per-control reference for gathering compliance evidence on AWS. For each control in `../controls-map.md`, this guide tells you the AWS console path, the CLI command, and the canonical export that satisfies an auditor or customer security review.

1. Access controls

MFA enforcement (control 1.1)

Console: IAM → Account settings → Multi-factor authentication status (for root). For users: IAM → Users → filter by `MFA active = No`. For Identity Center / IAM Identity Center: Identity Center → Settings → Authentication → MFA.

CLI:

```
# All users + their MFA status
aws iam list-users --query 'Users[].[UserName,Arn]' --output table

# Per-user MFA devices
for user in $(aws iam list-users --query 'Users[].UserName' --output text); do
  echo "$user: $(aws iam list-mfa-devices --user-name "$user" --query 'MFADevices[].SerialN
done

# Credentials report (most comprehensive — includes MFA flag per user)
aws iam generate-credential-report
aws iam get-credential-report --query Content --output text | base64 -d > credential-repor
```

Evidence to save: the credential report CSV — it's the single-page proof that lists every user, last activity, and MFA status.

IAM roles and least privilege (control 1.2)

Console: IAM → Roles → review each role's policies + trust relationship.

CLI:

```
# Full IAM authorization details for the account
aws iam get-account-authorization-details > iam-authz-details.json

# Just policies attached to a specific role
aws iam list-attached-role-policies --role-name <role>
aws iam list-role-policies --role-name <role>  # inline
```

Evidence: `iam-authz-details.json`. This is comprehensive but large; for a digestible version, also keep a written summary of the privileged roles.

Service-account / access-key inventory (control 1.3)

CLI:

```
# All IAM users + their access keys
for user in $(aws iam list-users --query 'Users[].UserName' --output text); do
  aws iam list-access-keys --user-name "$user"
done > access-key-inventory.json

# Stale keys (>90 days)
aws iam get-credential-report \
  --query Content --output text | base64 -d | \
  awk -F, 'NR==1 || ($11 < "'$(date -v-90d +%Y-%m-%d)'" && $10 == "true") { print }'
```

Evidence: `access-key-inventory-<date>.json` + documented rotation policy.

Better long-term: replace IAM users + access keys with IAM Identity Center (SSO) for humans and IAM Roles for Service Accounts (EKS/Lambda/ECS service-linked roles) for workloads.

2. Logging and monitoring

CloudTrail organization trail (control 2.1)

Console: CloudTrail → Trails → look for `IsOrganizationTrail = Yes`, `IsMultiRegionTrail = Yes`.

CLI:

```
# Confirm an org-level multi-region trail exists
aws cloudtrail describe-trails \
  --query 'trailList[?IsOrganizationTrail==`true` && IsMultiRegionTrail==`true`]'

# Confirm it's actively logging
aws cloudtrail get-trail-status --name <trail-name>

# Confirm management + data events are configured
aws cloudtrail get-event-selectors --trail-name <trail-name>
```

Evidence: `cloudtrail-org-config.json` + `cloudtrail-status.json`.

CloudTrail retention + tamper resistance (control 2.2 + 2.5)

CLI:

```
# S3 bucket lifecycle (where CloudTrail logs land)
aws s3api get-bucket-lifecycle-configuration --bucket <cloudtrail-bucket>

# Object Lock (tamper resistance)
aws s3api get-object-lock-configuration --bucket <cloudtrail-bucket>

# Sample query for oldest log (via Athena over CloudTrail)
# Replace <db> and <table> with your Athena CloudTrail config
# SELECT MIN(eventtime) FROM <db>.<table>;
```

Evidence: lifecycle policy + object-lock configuration. Use Object Lock in **Compliance** mode (not Governance) for stronger tamper resistance — even the root user can't override.

GuardDuty / Security Hub (control 2.4)

Optional but credibility-boosting:

```
# GuardDuty: enabled + active in each region
aws guardduty list-detectors
aws guardduty get-detector --detector-id <id>

# Security Hub findings
aws securityhub describe-hub
aws securityhub get-findings --max-results 10
```

4. Vulnerability and image scanning

ECR image scanning (control 4.2)

CLI:

```
# Per-repo scanning setting
aws ecr describe-image-scanning-configuration --repository-name <repo>

# Enable scan-on-push (one-time)
aws ecr put-image-scanning-configuration \
  --repository-name <repo> \
  --image-scanning-configuration scanOnPush=true

# Scan results for a specific image
aws ecr describe-image-scan-findings \
  --repository-name <repo> \
  --image-id imageDigest=<digest>
```

Better: use ECR Enhanced Scanning (Inspector v2) for continuous, deeper scans.

5. Backups, recovery

RDS automated backups (control 5.1)

CLI:

```
aws rds describe-db-instances --db-instance-identifier <id> \
  --query 'DBInstances[0].{BackupRetentionPeriod:BackupRetentionPeriod,PreferredBackupWindo

# Backup history
aws rds describe-db-snapshots \
  --db-instance-identifier <id> \
  --snapshot-type automated --max-records 30
```

Evidence: backup-retention config + last 7 days of successful snapshots.

AWS Backup (multi-service)

If using AWS Backup:

```
aws backup list-backup-plans
aws backup get-backup-plan --backup-plan-id <id>
aws backup list-recovery-points-by-backup-vault --backup-vault-name <vault>
```

6. AI workload controls

If running on AWS Bedrock:

```
# Per-account model access policies
aws bedrock list-foundation-models

# Bedrock guardrails (content filtering, PII detection)
aws bedrock list-guardrails
aws bedrock get-guardrail --guardrail-identifier <id>
```

```
# CloudWatch billing alarm for Bedrock spend
aws cloudwatch describe-alarms --alarm-name-prefix "bedrock-spend"
```

Authoritative references

- AWS Well-Architected Security Pillar: https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html
- AWS Foundational Security Best Practices (Security Hub): https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-standards-fsbp.html
- CIS AWS Foundations Benchmark: https://www.cisecurity.org/benchmark/amazon_web_services
- AWS Audit Manager (for SOC 2, ISO 27001 evidence mapping): https://docs.aws.amazon.com/audit-manager/

platforms/azure.md

Azure evidence guide

Per-control reference for gathering compliance evidence on Microsoft Azure.

1. Access controls

MFA enforcement (control 1.1)

Console: Entra ID → Security → Conditional Access → look for a policy requiring MFA on the relevant role/group.

CLI:

```
# Conditional Access policies
az ad signed-in-user list-owned-objects --query "[?@.['@odata.type']=='#microsoft.graph.po

# Better via Microsoft Graph
az rest --method GET --url https://graph.microsoft.com/v1.0/identity/conditionalAccess/pol

# Per-user MFA status — Sign-in logs filtered
az monitor activity-log list --max-events 100 --filters "category=Sign-in"
```

Evidence: the Conditional Access policy export — JSON of the rule requiring MFA — plus a sample sign-in log entry showing MFA was satisfied.

Privileged Identity Management (PIM)

If using PIM (recommended for production), evidence is the eligible vs active assignment list:

```
# Eligible role assignments
az rest --method GET --url "https://graph.microsoft.com/v1.0/roleManagement/directory/roleI
```

```
# Active role assignments
az rest --method GET --url "https://graph.microsoft.com/v1.0/roleManagement/directory/role
```

PIM-based just-in-time elevation is strong evidence on its own.

Service principals + secrets (control 1.3)

```
# All service principals (filter to app SPs, not enterprise apps)
az ad sp list --filter "servicePrincipalType eq 'Application'" --query "[].{appId:appId, di

# Credentials per service principal
az ad sp credential list --id <sp-id>
```

Better long-term: Managed Identities (no secrets) for Azure workloads; OIDC federation for external CI/CD.

2. Logging and monitoring

Azure Activity Log + Diagnostic Settings (control 2.1)

Subscription-level Activity Log is on by default; what you need to verify is that it's exported to durable storage:

```
# Diagnostic settings on a subscription
az monitor diagnostic-settings subscription list \
  --subscription <subscription-id>

# Diagnostic settings on a specific resource
az monitor diagnostic-settings list --resource <resource-id>
```

For org-wide capture, use Azure Policy with the "Audit Diagnostic Settings" policy and enforce it across all subscriptions in a management group.

Retention (control 2.2)

If exporting to a Storage Account:

```
az storage account blob-service-properties show \
  --account-name <storage-account>

# Retention policies (legal hold + time-based)
az storage container immutability-policy show \
  --account-name <storage-account> --container <container>
```

For Log Analytics workspace retention:

```
az monitor log-analytics workspace show \
  --resource-group <rg> --workspace-name <name> \
  --query "retentionInDays"
```

Microsoft Defender for Cloud (control 2.4 + 4.x)

The Azure equivalent of GuardDuty / SCC. Enable the plans you need (Servers, Containers, Storage, etc.) and use Defender recommendations as compliance evidence.

```
az security pricing list
az security alert list
```

3. Change management

For ARM / Bicep deployments, the deployment history is your audit trail:

```
az deployment sub list --query "[].{name:name, timestamp:properties.timestamp, principalNa
```

For GitHub-based deploys, see `github.md`.

5. Backups, recovery

Azure SQL backups (control 5.1)

```
# Long-term retention configuration
az sql db ltr-policy show --resource-group <rg> --server <server> --name <db>

# Short-term (point-in-time recovery)
az sql db show --resource-group <rg> --server <server> --name <db> \
  --query "currentBackupStorageRedundancy"

# List available backups
az sql db ltr-backup list --resource-group <rg> --server <server> --database <db>
```

Azure Backup (multi-service)

```
az backup vault list
az backup item list --resource-group <rg> --vault-name <vault>
az backup recoverypoint list --resource-group <rg> --vault-name <vault> --container-name <
```

6. AI workload controls

Azure OpenAI (control 6.1, 6.3)

```
# Resource-level network restrictions
az cognitiveservices account show \
  --name <resource> --resource-group <rg> \
  --query "properties.networkAcls"

# Per-deployment quota
```

```
az cognitiveservices account deployment list \
  --name <resource> --resource-group <rg>
```

Azure OpenAI content filters are configured per-deployment: evidence is the deployment's `raiPolicy` setting.

Azure budget alerts

```
az consumption budget list --resource-group <rg>
```

Authoritative references

- Azure Security Benchmark v3: https://learn.microsoft.com/en-us/security/benchmark/azure/overview
- Microsoft Cloud Security Benchmark: https://learn.microsoft.com/en-us/security/benchmark/cloud/
- CIS Microsoft Azure Foundations Benchmark: https://www.cisecurity.org/benchmark/azure
- Azure Policy compliance: https://learn.microsoft.com/en-us/azure/governance/policy/concepts/regulatory-compliance
- Azure OpenAI responsible AI: https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/content-filter

platforms/gcp.md

GCP evidence guide

Per-control reference for gathering compliance evidence on Google Cloud Platform.

1. Access controls

MFA enforcement (control 1.1)

Workspace admin console: Admin → Security → 2-step verification → Enforcement. Cloud Identity admins are tied to Workspace; enforce 2SV there.

CLI (Cloud Identity / Workspace via gcloud + Admin SDK):

```
# Users with 2SV status
gcloud admin users list --filter="isEnrolledIn2Sv=true" --format=json
```

For dedicated Cloud Identity (no Workspace), the Admin SDK API is the source of truth.

IAM roles + least privilege (control 1.2)

CLI:

```
# Project-level IAM policy
gcloud projects get-iam-policy <project-id> --format=json > iam-policy.json

# Org-level IAM
gcloud organizations get-iam-policy <org-id> --format=json > org-iam-policy.json

# Asset Inventory: comprehensive snapshot of all IAM across projects
gcloud asset search-all-iam-policies --scope=organizations/<org-id> > org-iam-asset.json
```

Evidence: the asset-inventory IAM export is the most comprehensive single artifact.

Service-account keys (control 1.3)

CLI:

```
# All service accounts in a project
gcloud iam service-accounts list --project=<project>

# Keys per service account (look for USER_MANAGED — those are long-lived)
for sa in $(gcloud iam service-accounts list --project=<project> --format='value(email)');
  gcloud iam service-accounts keys list --iam-account="$sa" \
    --filter="keyType=USER_MANAGED" --format=json
done > user-managed-keys.json
```

The right answer is Workload Identity Federation, not key rotation. WIF lets workloads (GKE, Cloud Run, Cloud Build, even external systems via OIDC) authenticate without long-lived keys.

```
# Inventory of workload identity pools
gcloud iam workload-identity-pools list --location=global
```

2. Logging and monitoring

Cloud Audit Logs sink (control 2.1)

Console: Logging → Log Router → look for an org-level sink exporting to BigQuery, GCS, or Pub/Sub.

CLI:

```
# Org-level sinks
gcloud logging sinks list --organization=<org-id>
```

```
gcloud logging sinks describe <sink-name> --organization=<org-id>

# Data Access logs configuration per project (these are NOT on by default)
gcloud projects get-iam-policy <project> --format='value(auditConfigs)'
```

Important: Admin Activity and System Event logs are always on and free. **Data Access logs are off by default** and must be explicitly enabled. Customer security reviews ask about Data Access logs specifically.

Retention + tamper resistance (control 2.2 + 2.5)

For logs sinked to GCS:

```
# Bucket retention policy
gsutil retention get gs://<bucket>

# Retention lock — once locked, even admins can't shorten retention
# Lock it after confirming the retention is right
gsutil retention lock gs://<bucket>
```

For logs sinked to BigQuery: set table partitioning expiration to your retention window.

Cloud Monitoring alerts (control 2.4)

```
# Alert policies
gcloud alpha monitoring policies list --format=json

# Notification channels (where do alerts go?)
gcloud beta monitoring channels list
```

Evidence: alert policies + notification channels mapped to your on-call rotation.

4. Vulnerability scanning

Container Analysis (control 4.2)

Enable:

```
gcloud services enable containeranalysis.googleapis.com
gcloud services enable containerscanning.googleapis.com
```

Query results:

```
# Vulnerabilities for a specific image
gcloud artifacts docker images describe <image-uri> --show-package-vulnerability

# Org-wide via Asset Inventory
gcloud asset search-all-resources \
  --scope=organizations/<org-id> \
  --asset-types=containeranalysis.googleapis.com/Occurrence
```

Security Command Center (control 4.x + 2.4)

GCP's equivalent of AWS Security Hub. Enable Standard tier (free) or Premium for richer scanning. Findings are credible evidence on their own.

```
gcloud scc findings list <org-id>
```

5. Backups, recovery

Cloud SQL backups (control 5.1)

```
# Backup config
gcloud sql instances describe <instance> \
  --format='yaml(settings.backupConfiguration)'
```

```
# Backup history (last 10)
gcloud sql backups list --instance=<instance> --limit=10
```

Cloud Storage versioning + object lifecycle (control 5.1)

```
gsutil versioning get gs://<bucket>
gsutil lifecycle get gs://<bucket>
```

6. AI workload controls

Vertex AI quotas + endpoints (control 6.1, 6.3)

```
# List deployed models
gcloud ai endpoints list --region=<region>
gcloud ai models list --region=<region>

# Endpoint config — confirm auth is required
gcloud ai endpoints describe <endpoint-id> --region=<region>

# Per-project quotas
gcloud services quota list --service=aiplatform.googleapis.com --consumer=projects/<projec
```

Vertex AI Safety Filters (for generative models): apply them at deploy time and document the configuration.

Cloud Billing budgets + spend caps

```
# Billing budgets
gcloud billing budgets list --billing-account=<id>
```

Set budget alerts at 50/80/100% of monthly cap. Hard caps require custom code (Pub/Sub → Cloud Function to disable billing).

Authoritative references

- Google Cloud Security Best Practices: https://cloud.google.com/security/best-practices
- CIS Google Cloud Platform Foundations Benchmark: https://www.cisecurity.org/benchmark/google_cloud_computing_platform
- Cloud Audit Logs overview: https://cloud.google.com/logging/docs/audit
- Compliance Reports Manager: https://cloud.google.com/security/compliance
- Vertex AI Responsible AI guidelines: https://cloud.google.com/vertex-ai/docs/generative-ai/learn/responsible-ai

platforms/github.md

GitHub evidence guide

Per-control reference for gathering compliance evidence in GitHub. Most engineering-led SOC 2 readiness work has GitHub as the critical-control surface for code review, deploy approvals, and secret scanning.

1. Access controls

Org membership + MFA (control 1.1)

UI: Org → People → Outside collaborators tab. Org → Settings → Authentication security → "Require two-factor authentication".

CLI:

```
# Org-level 2FA enforcement
gh api orgs/<org> --jq '.two_factor_requirement_enabled'

# Members with 2FA disabled (requires admin scope)
gh api "orgs/<org>/members?filter=2fa_disabled" --jq '.[].login'

# Org owners
gh api orgs/<org>/members --jq 'map(select(.role=="admin"))'
```

Evidence: confirm `two_factor_requirement_enabled = true` org-wide. This is the right control; it prevents anyone from being a member without 2FA.

Personal Access Tokens

PATs are a common blind spot. Even with 2FA, a leaked PAT lets an attacker bypass it.

```
# Audit log entries for PAT creation (Enterprise only)
gh api orgs/<org>/audit-log --jq 'map(select(.action == "user.token_creation"))'
```

```
# OAuth apps + GitHub Apps installed on the org
gh api orgs/<org>/installations --jq '.installations[].account.login'
```

Best practice: replace PATs with GitHub Apps (org-installable, auditable, fine-grained permissions). Document any remaining PATs with owner + rotation date.

CODEOWNERS (control 3.1 dependency)

```
# Pull the canonical CODEOWNERS
gh api repos/<owner>/<repo>/contents/.github/CODEOWNERS \
  --jq '.content' | base64 -d
```

3. Change management

Branch protection / Rulesets (control 3.1)

GitHub has two systems: legacy "Branch Protection" and newer "Rulesets" (more flexible, org-applicable). Rulesets are the direction; both are auditor-acceptable.

```
# Legacy branch protection
gh api repos/<owner>/<repo>/branches/main/protection > branch-protection-main.json

# Newer rulesets
gh api repos/<owner>/<repo>/rulesets > rulesets.json
gh api repos/<owner>/<repo>/rulesets/<id> > ruleset-detail.json

# Org-level rulesets (apply to all repos in the org)
gh api orgs/<org>/rulesets > org-rulesets.json
```

Org-level rulesets are the strongest evidence — they apply the same protection across every repository, so adding a new repo doesn't accidentally create an unprotected production branch.

Environment protection (control 3.2)

```
# Environments on a repo
gh api repos/<owner>/<repo>/environments > environments.json

# Production env detail (required reviewers, deployment branch
# restrictions, secrets scope)
gh api repos/<owner>/<repo>/environments/production > env-production.json
gh api repos/<owner>/<repo>/environments/production/deployment-protection-rules > prod-pro
```

Deploy history (control 3.4)

```
# Last N workflow runs on main
gh run list --workflow=deploy.yml --branch=main --limit=30 \
  --json databaseId,displayTitle,headSha,createdAt,actor,conclusion \
  > recent-deploys.json

# Per-deploy approver
gh api repos/<owner>/<repo>/actions/runs/<id>/approvals
```

Signed commits (bonus credibility)

```
# Branch protection rule: require signed commits
gh api repos/<owner>/<repo>/branches/main/protection \
  --jq '.required_signatures.enabled'

# Verify a specific commit
gh api repos/<owner>/<repo>/commits/<sha> --jq '.commit.verification'
```

4. Vulnerability + secret scanning

Dependabot (control 4.1)

```
# Dependabot config in repo
gh api repos/<owner>/<repo>/contents/.github/dependabot.yml \
  --jq '.content' | base64 -d

# Dependabot alerts
gh api repos/<owner>/<repo>/dependabot/alerts \
  --jq '[.[] | {number, severity, state, package: .dependency.package.name, fix_resolved: .f
  > dependabot-alerts.json
```

Secret scanning + push protection (control 4.3)

```
# Org-wide secret scanning + push protection
gh api orgs/<org> --jq '.security_and_analysis'

# Per-repo
gh api repos/<owner>/<repo> --jq '.security_and_analysis'

# Open alerts
gh api repos/<owner>/<repo>/secret-scanning/alerts \
  --jq '[.[] | {number, secret_type, state, resolution, created_at}]'
```

The control that matters: `secret_scanning_push_protection` set to `enabled` org-wide. Push protection prevents the secret from reaching the repo in the first place.

Code scanning (CodeQL / SAST)

```
# Enable
gh api repos/<owner>/<repo>/code-scanning/default-setup -X PUT \
  --field state=configured

# Alerts
gh api repos/<owner>/<repo>/code-scanning/alerts \
  --jq '[.[] | {number, rule_id, severity, state}]'
```

Audit log

GitHub Enterprise / GHE Cloud has an org-level audit log API. Pull this quarterly for the compliance evidence trail:

```
gh api "orgs/<org>/audit-log?phrase=created:>2026-04-01" \
  --paginate > audit-log-2026-Q2.json
```

This is gold for "who did what" reconstruction during an incident.

Authoritative references

- GitHub security best practices: https://docs.github.com/en/code-security/getting-started/github-security-features
- GitHub Advanced Security: https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security
- Branch protection rules: https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/about-protected-branches
- Rulesets: https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-rulesets/about-rulesets
- Secret scanning + push protection: https://docs.github.com/en/code-security/secret-scanning/introduction/about-secret-scanning

NEXT STEPS

Want an engineer to do the work?

This kit gives you the scaffold; a Controls Review goes further — 48 hours inspecting your actual systems, producing a written findings report with severity, evidence requested, fix path, and effort estimate. The follow-on Controls Sprint ships merged fixes and a populated evidence folder. Book a 15-minute fit call or reach out directly.

Book a controls review →

or email **hello@musabdulai.com**