obsinto.

FREE EDITION

ENGINEERING GUIDANCE

# Secure Software Development Guidelines

## Free Edition

Eighteen engineering practices for teams building SaaS products. Each one makes your product safer now, and each one quietly produces the evidence that SOC 2, ISO 27001, HIPAA and PCI DSS will ask you for later.

Security by Design · Compliance Ready · Cloud First


| DOCUMENT | VERSION | ISSUED | AUDIENCE |
|---|---|---|---|
| OBS-ENG-SSDG-001 | 1.0 | August 2026 | Founders, CTOs, engineers |

PURPOSE

# Compliance is downstream of architecture

Most teams meet compliance the painful way. They build fast for two years, land an enterprise deal that requires SOC 2, and then discover that the audit is not really about paperwork. It is about whether the product was built with certain habits, and whether anyone can prove it.

The paperwork part is genuinely quick. What actually delays certification is the stuff you cannot backfill: audit history that was never collected, tenant isolation that has to be retrofitted into a live database, an access model nobody wrote down. Those are engineering problems, and by the time an auditor finds them they are expensive engineering problems.

So this guide takes the opposite approach. Build with these eighteen practices from the start and the certification largely assembles itself. None of them exist for the auditor's benefit. They are just good engineering that happens to leave a paper trail.

## Where to start, based on where you are

| Your stage | Do these first | Why |
|---|---|---|
| **Designing the schema** | 1, 2, 3, 4 | Data classification and the access model are the hardest things to change later. |
| **Building toward first customers** | 5 through 10 | Cheap to set up now. Each becomes an audit finding if you skip it. |
| **Live, with paying customers** | 11, 12, 13, 16 | You now carry availability and privacy obligations whether you planned to or not. |
| **Preparing for a first audit** | 14, 15, 17 | Auditors test process and documentation, not just configuration. |

Each practice ends with a green panel naming the artifacts it produces as a side effect. Those artifacts are exactly what an auditor will request. If you keep nothing else from this guide, keep that framing: **a control that leaves no artifact behind cannot be verified, by an auditor or by you.**

NAVIGATION

# Contents

PART

# 01 Data, identity and access

Three decisions that are cheap this week and brutal to change once you have customers.

IN THIS PART

# 1 Data governance

Know what you collect before you collect it.

Every field you store is a field you have to protect, justify, retain and eventually delete. Data collected "because we might want it later" carries the full cost of protection and none of the value. Before adding a column, ask what product decision depends on it.

- **Classify everything.** Four tiers is plenty: public, internal, confidential, regulated. Let the tier drive the handling rules so you never decide case by case.
- **Write a data inventory.** For each element: what it is, why you have it, who owns it, where it lives, how long you keep it, how it dies. A spreadsheet is fine.
- **Define retention when you add the field**, even if the answer is seven years. Deciding is the point.

⚠ COMMON MISTAKE

**Retention defined as "forever, probably".**
Teams define collection and never define deletion. "We have never deleted anything" costs you a finding under SOC 2, ISO 27001 and GDPR alike.

EVIDENCE THIS PRODUCES

- A data inventory.
- A classification policy.
- A retention schedule.

# 2 Authentication and identity

The one area where you should almost never write it yourself.

- **Use an established identity provider.** The primitives are subtle and the failure modes are severe. If you sell to businesses, support single sign-on early; it is the first thing enterprise buyers ask about.
- **If you store passwords, hash them with Argon2 or bcrypt.** Per-user salt, a cost factor you revisit. Never plain SHA, never anything reversible.
- **MFA on every privileged path.** Not just your admin console. Also the cloud account, the CI system, the source repo, the DNS registrar.
- **Manage sessions deliberately.** Absolute expiry plus idle timeout, Secure and HttpOnly cookies, server-side invalidation on logout and password change.
- **Rate-limit login attempts** per account and per source. Credential stuffing is automated and cheap.

⚠ COMMON MISTAKE

**MFA on the console, a long-lived API token beside it.**
A protected login is worth little if a static token with the same permissions sits in a config file. Inventory non-human credentials with the same seriousness as user accounts. Scope them narrowly and give them an expiry.

EVIDENCE THIS PRODUCES

- Identity provider configuration export.
- A privileged account list with second factors.
- Session policy and tests.

# 3 Authorization

Where most real breaches in multi-tenant products actually happen.

Authentication says who is calling. Authorization decides what they may do, and it has to be right on every path, not just at the front door.

- **A few roles, least privilege each.** Resist per-user permission flags. They multiply until nobody can reason about the matrix.
- **Enforce on the server, always.** Hiding a button is a courtesy to honest users. Anyone else has the developer console open.
- **Deny by default.** A route with no declared permission should fail closed, so the endpoint a new engineer forgets about refuses to serve rather than serving everything.

THE PRACTICE

**Derive the tenant from the session, never from the request.**
In a multi-tenant product, the tenant identifier used to scope a query comes from the authenticated session. Never from a URL parameter, header or request body that a caller can edit. Enforce it in one shared data-access layer, or in the database with row-level security, so a forgotten filter returns nothing instead of returning another company's records.

COMMON MISTAKE

**Testing only the happy path.**
Write the negative tests: a viewer cannot write, tenant A gets a 404 for tenant B's record id (a 403 confirms the record exists), a guessed sequential id fails.

EVIDENCE THIS PRODUCES

- A role and permission matrix.
- Automated cross-tenant and cross-role denial tests.
- Access review records.

PART

# 02 Platform and infrastructure

Where your code runs, what it holds, and what it records.

IN THIS PART

# 4 Environment separation

Limits how far one mistake can travel.

- **Separate cloud accounts, not namespaces.** Account boundaries are the one boundary your provider enforces for you. An over-broad permission inside a shared account reaches everything.
- **Separate credentials.** A leaked development key must be worthless against production.
- **Separate data.** Lower environments get synthetic fixtures, never a copy of the production database.

THE PRACTICE

**Never routinely test with production data.**
Build a fixture generator that produces realistic synthetic records. The moment production data lands in a dev environment, that environment inherits every obligation production has, and it almost certainly lacks the controls to match.

COMMON MISTAKE

**The "anonymized" production copy.**
Masking done under deadline pressure is nearly always reversible: names get scrambled while free-text fields, uploads and identifiers survive. Treat a masked dump as production data unless you can prove otherwise.

EVIDENCE THIS PRODUCES

- An inventory of cloud accounts and what each is for.
- Distinct credentials per environment.
- A no-production-data rule, ideally enforced.

# 5 Secrets management

The most common serious finding in early codebases, and the most avoidable.

- **Use a real secrets manager.** Every major cloud has one, and workload identity means your app never holds a bootstrap credential.
- **Scan for secrets in CI and in history.** Pre-commit hook plus pipeline check. Scan the full git history once when you adopt it; the interesting findings are usually old.
- **One job per credential.** Separate read from write, per service, with an expiry. Never one shared key used by every service and every engineer.
- **Rotate on schedule and on events:** suspected exposure, and whenever someone with access leaves.

THE PRACTICE

**Treat any committed secret as compromised.**
Rotate first, clean up the commit second. Deleting the commit is not remediation; assume the value was exposed the entire time it was there, because private repos get cloned, forked and backed up in ways nobody tracks.

EVIDENCE THIS PRODUCES

- A secrets inventory showing what is stored and who can read it.
- Secret-scanning results from the pipeline.
- A rotation record with dates and triggers.

# 6 Encryption

The risk is not weak algorithms. It is unencrypted things you forgot about.

- **TLS 1.2+ on every public endpoint**, HTTP redirected, HSTS on. Run an external scan to confirm.
- **TLS internally too**, service to service and app to database, not just at the edge.
- **Managed encryption at rest** on databases, object storage, queues, log stores. Flip it on at creation; it rarely inherits downstream.
- **Managed keys, split permissions.** Who can use a key and who can administer it are different questions. Answer them separately.

⚠ COMMON MISTAKE

**The primary database is encrypted; its backups are not.**
Snapshots, exports, cross-region replicas, log archives and analytics copies each need checking. An auditor will ask about backups specifically.

EVIDENCE THIS PRODUCES

- An external TLS configuration scan.
- Encryption-at-rest settings for every data store, including backups.
- Key policies showing who may use and who may administer.

# 7 Audit logging

Worth almost nothing if you start collecting after you need it.

Log five classes of event and you cover what incidents and audits ask about: authentication (success and failure), administrative actions, permission changes, configuration changes, and data exports.

- **Structured, not prose**, so you can query instead of grepping under pressure.
- **Centralized and append-only.** Logs that live only on the machine that made them vanish exactly when you need them. Operators should not be able to edit history.
- **UTC everywhere.** Correlating across services needs one clock.
- **Pick a retention period**, usually a year for audit purposes, and configure it on purpose.

⚠ COMMON MISTAKE

**Logging the request body, and with it the personal data.**
Audit logs quietly become the least protected copy of your most sensitive information. Log identifiers and metadata, never credentials, document content or full personal records. Add a test, because it creeps back with every debug statement.

EVIDENCE THIS PRODUCES

- Log retention configuration.
- A sample extract showing the fields captured per event class.
- Evidence that logs are immutable and centrally held.

# 8 API security

Anything your web client can do, a script can do faster and without the guardrails.

- **Authenticate every request.** Health checks should not enumerate internals.
- **Authorize every action**, per request, on the specific object touched. Once at the gateway is not enough.
- **Validate input against an explicit schema** and reject what fails, rather than sanitizing and hoping. Size limits on bodies, arrays and uploads.
- **Rate-limit**, tighter on login, password reset, invitations and exports.
- **Serialize explicit fields.** Serializers that default to "everything" leak new columns the day they are added.

⚠ COMMON MISTAKE

**Sequential ids plus a missing ownership check.**
This is the most common serious API vulnerability in the wild. If your ids are guessable integers and any handler resolves the object before checking ownership, someone will find it by counting.

EVIDENCE THIS PRODUCES

- An API specification showing authentication and permissions per endpoint.
- Rate-limiting configuration.
- Automated tests covering unauthenticated and unauthorized access per route.

PART

# 03 Code, dependencies and cloud

What you write, what you borrow, and what you run it on.

IN THIS PART

# 9 Secure coding

The vulnerability classes that matter have been stable for twenty years.

Follow OWASP guidance and prefer framework defaults over clever code. The frameworks have already had the bugs you are about to write.

- **Injection:** parameterized queries or the ORM builder, never string concatenation into a query.
- **XSS:** let the template engine encode output. Any "render raw HTML" call gets reviewed.
- **CSRF:** framework tokens plus SameSite cookies.
- **Deserialization:** data formats, not object formats, for anything untrusted.
- **Errors:** generic message to the client, detail to the log. No stack traces in responses.

THE PRACTICE

**Automate what you can, review what you cannot.**
Static analysis on every commit, blocking. It will not catch logic flaws or missing authorization checks, which is exactly what human review is for. Spend the machine on mechanical classes and the human attention on the rest.

EVIDENCE THIS PRODUCES

- A written coding standard, versioned with the code.
- Static analysis results per build.
- Code review records showing an independent reviewer.

# 10 Dependency management

Most of your product was written by someone else. You are responsible for it anyway.

- **Adopt deliberately.** Is it maintained, does the license work for you, does it respond to security reports, how much transitive weight does it drag in.
- **Commit lockfiles** so builds are reproducible and a scan result means something.
- **Scan on every commit and on a daily schedule.** A vulnerability published tomorrow affects the dependency you did not touch today.
- **Patch targets by severity**, and hold them: critical in days, high in weeks, the rest on a planned cadence.
- **Remove what you stopped using.** Same risk, zero benefit.

> ⚠ COMMON MISTAKE
>
> **Scanning the app and ignoring the base image.**
> Containers ship an operating system with its own vulnerabilities, and build caching can pin a stale layer for months. Scan built images and rebuild bases on a schedule.

> EVIDENCE THIS PRODUCES
>
> - A software bill of materials per build.
> - Scan results, per build and per scheduled scan.
> - A remediation record showing findings closed within target.

# 11 Cloud security

The provider secures the infrastructure. You secure the configuration.

- **Lock the root account:** MFA, no access keys, no day-to-day use. Work through named users and roles.
- **No standing admin.** Elevated access is assumed for a purpose and expires.
- **Provider audit logging on**, every region, delivered somewhere the account's own admins cannot quietly edit.
- **Closed network by default:** no public databases, no open management ports, no public buckets unless they serve genuinely public assets.
- **Infrastructure as code**, reviewed and diffable. Console changes are invisible to your change process and are where drift starts.

**THE PRACTICE**

**Run a configuration benchmark today.**
Every provider has a posture tool and open-source scanners exist for all of them. The first run takes an hour and typically finds a public snapshot, an over-permissive role, and a region with logging off. Then schedule it.

**EVIDENCE THIS PRODUCES**

- Benchmark or posture scan results, with remediation history.
- An infrastructure-as-code repository showing reviewed changes.
- Provider audit logging configuration and its retention.

PART

# 04 Resilience, privacy and lifecycle

Obligations that arrive with your first real customer.

IN THIS PART

# 12 Operational resilience

Decide your recovery targets before an incident decides them for you.

- **Pick an RTO and RPO** and write them down. For an early B2B product, four hours down and fifteen minutes of data loss are defensible starting answers.
- **Back up daily, keep 30 days, store a copy in another region.** Point-in-time recovery on a managed database gets you most of the way.
- **Health checks that reflect real dependencies**, alerts routed to a person who agreed to be woken, and a written incident procedure with severity levels and customer communication times.

⚠ COMMON MISTAKE

**Backups that have never been restored.**
An untested backup is a hypothesis. The job silently stopped weeks ago, the snapshot missed the volume that matters, the old encryption key is gone, or the restore takes eleven hours against a four-hour target. You find these in a quarterly rehearsal or you find them in production.

EVIDENCE THIS PRODUCES

- A business continuity and disaster recovery plan naming RTO and RPO.
- Restore test records with dates and outcomes.
- Alert definitions, routing, and incident records.

# 13 Privacy by design

Knowing where personal data is turns out to be most of the work.

Three questions, answered in writing. What personal data do you process, including logs, analytics, support tickets and whatever customers upload into your product? Why, in one sentence per purpose? And how will you delete it, on request and on schedule, including from backups and every third party you shared it with?

THE PRACTICE

**Build the deletion path while you build the create path.**
Once personal data has fanned out to a search index, a warehouse, a support tool, an email platform and thirty days of backups, honoring one deletion request becomes a project. Design the fan-out and the fan-in together, and keep a register of every system that receives personal data.

- **Join on internal ids, not email addresses**, so pseudonymization is possible later.
- **Keep a sub-processor list.** Customers will ask and contracts will require it.
- **Name an owner for data subject requests** before the first one arrives. The deadlines are statutory.

EVIDENCE THIS PRODUCES

- A record of processing activities.
- A data subject request procedure.
- A sub-processor register.

## 14 Secure SDLC

Security inside how you already work costs very little. As a separate phase, it costs a lot and gets dropped.

The minimum viable version is four rules, each about a day to set up:

1. Every change is peer reviewed, enforced by branch protection with self-approval off.
2. Automated tests run on every change and block the merge on failure.
3. Dependency and secret scanning run on every change, also blocking.
4. Nothing reaches production without passing all three.

THE PRACTICE

**Why this one matters most.**
Branch protection plus pull request history is the single most useful evidence artifact a young company can have. It demonstrates change authorization, segregation of duties and testing at once, three separate control areas, and it accumulates automatically from work you were doing anyway. It also cannot be created retrospectively. Turn it on this week.

**EVIDENCE THIS PRODUCES**

- Branch protection settings, exportable as point-in-time evidence.
- Pull request records showing reviewer and approval time.
- Pipeline definitions showing which checks block a merge.

PART

# 05 Process and proof

The habits that make a well-built product demonstrably well-built.

IN THIS PART

# 15 Change management

For most SaaS teams this is your existing PR workflow with the gates actually enforced.

- **Everything in version control:** app code, infrastructure, migrations, pipeline config. If it can change production and is not in a repo, it is outside your process.
- **Review before merge, by someone other than the author.**
- **Track what shipped:** a deployment record linking the version to its changes and approver.
- **Know the rollback**, especially with database migrations. Write migrations backward-compatible with the running version and run them before the new code takes traffic, so a failure leaves the old version serving.

THE PRACTICE

**Make the emergency path a shorter route, not a bypass.**
You will need to ship urgently one day. Decide now that urgent changes still get reviewed, tested and recorded, just faster, with a retrospective look afterwards. A documented expedited procedure is a control. An undocumented override is a finding.

EVIDENCE THIS PRODUCES

- Deployment records with version, approver and outcome.
- An emergency change register with retrospective reviews.

# 16 Continuous monitoring

Turns "we configured it once" into "we can show it still works".

Five things to watch: system health (your customers should not be your monitoring), authentication failures (the visible signature of credential attacks), privilege and configuration changes, infrastructure alarms including certificate expiry and backup job failure, and data access far outside a principal's normal pattern.

⚠ COMMON MISTAKE

**Alerts nobody reads.**
A channel with a hundred daily messages is the same as no monitoring, except you believe you are covered. Every alert gets a defined response; anything with no action attached becomes a dashboard metric instead.

**EVIDENCE THIS PRODUCES**

- Alert definitions and their routing.
- A sample of alerts with the response taken.
- Incident records, including the false alarms.

# 17 Architecture documentation

What you need for an audit is what you need for onboarding anyway.

Three artifacts. An architecture diagram: components, boundaries, where each runs, how they authenticate to each other, one page. A data-flow diagram showing where sensitive data enters, lives, moves and leaves; this one determines your audit scope, so accuracy here narrows the assessment. And a register of third-party integrations: what each receives, why, and where it operates.

**COMMON MISTAKE**

**A beautiful diagram, eighteen months stale.**
Old documentation is worse than none because people make decisions from it. Keep diagrams as text-based sources in the repo and update them in the same PR that changes the architecture.

**EVIDENCE THIS PRODUCES**

- A current architecture diagram, versioned.
- A data-flow diagram covering sensitive data.
- A third-party and sub-processor register.

# 18 Think long-term

Five switches to flip this week, because history cannot be backfilled.

First audits are rarely delayed by a missing policy document. Policies can be written in a week. They are delayed by absent history (an auditor testing six months of change control needs six months of records), by architectural retrofits (tenant isolation and deletion paths are engineering programs, not paperwork), and by nobody being able to say what the system actually does.

THE PRACTICE

**Enable the things that accrue history.**
Branch protection, provider audit logging, dependency scanning, secret scanning, backup restore checks. Each takes under a day. Each is worth more every week it runs.

EVIDENCE THIS PRODUCES

- Months of evidence, automatically.

## Key takeaway

> Do not stop building to do compliance. Build with these habits, and the compliance largely does itself.

# A Readiness checklist

Score each practice honestly, and only where you could hand an auditor the artifacts it produces. Doing it without being able to show it scores as not doing it, because that is how an auditor will score it too. The web edition at obsinto.com/resources computes weighted readiness against SOC 2, ISO 27001, HIPAA, PCI DSS and GDPR as you tick.

| Ref. | Practice | You can say yes when… | Yes? |
|---|---|---|---|
| §1 | Data governance | Inventory exists, classified, with retention and deletion defined | |
| §2 | Authentication | MFA on all privileged access; passwords hashed with a modern algorithm | |
| §3 | Authorization | Least-privilege roles; server-side enforcement; negative tests exist | |
| §4 | Environment separation | Separate accounts; no production data in lower environments | |
| §5 | Secrets management | Secrets in a manager; scanning in CI; rotation on schedule and events | |
| §6 | Encryption | TLS everywhere; encryption at rest including backups and snapshots | |
| §7 | Audit logging | Auth, admin, permission, configuration and export events logged centrally | |
| §8 | API security | Every endpoint authenticated and authorized; input validated; rate limits set | |
| §9 | Secure coding | Coding standard adopted; static analysis blocking on every commit | |
| §10 | Dependency management | Lockfiles committed; daily scans; patch targets met; SBOM produced | |
| §11 | Cloud security | Root locked down; audit logging on; network closed by default; IaC in use | |
| §12 | Operational resilience | RTO and RPO defined; backups running; a restore actually tested | |
| §13 | Privacy by design | Personal data mapped; deletion path built; sub-processor register kept | |
| §14 | Secure SDLC | Peer review, tests and scanning all enforced and blocking before merge | |
| §15 | Change management | Everything in version control; changes approved, tested, tracked | |
| §16 | Continuous monitoring | Alerts defined for health, auth failures, privilege change, infra | |
| §17 | Architecture documentation | Architecture and data-flow diagrams current; register maintained | |
| §18 | Think long-term | The five history-accruing controls above are enabled and running | |

# B References

Everything here is free to read, and worth reading in the original when you reach the part of your build it governs.

| Source | Read it when |
|---|---|
| **AICPA Trust Services Criteria** | Scoping a SOC 2, or answering a customer security questionnaire. |
| **NIST SP 800-218, Secure Software Development Framework** | Formalizing practices 14 and 15. |
| **NIST SP 800-207, Zero Trust Architecture** | Designing service-to-service authentication, practices 3 and 11. |
| **OWASP Application Security Verification Standard** | Turning practices 2, 3, 8 and 9 into testable acceptance criteria. |
| **OWASP Top 10** | Onboarding engineers; setting a static analysis baseline. |
| **OWASP API Security Top 10** | Working through practice 8. |
| **CISA Secure by Design** | Setting engineering direction, and practice 18. |

## About Obsinto

Obsinto is a compliance intelligence platform. It reads what your systems and documents actually show, writes the policies and evidence your framework requires from your own context, and keeps the picture current as your product changes. This guide is free because the practices in it are good engineering first and compliance second.

The web edition, with a live readiness self-assessment, lives at **obsinto.com/resources**. Questions or corrections: **support@obsinto.com**.

obsinto.

Compliance for the Age of Intelligence.

Building a product and wondering which of these to do first?

Questions or corrections **support@obsinto.com**

The web edition, with a live self-assessment **obsinto.com/resources**

On the web **obsinto.com**