Operations Guide

OrgGuardian customer guide — scans, findings, dashboards, alerts, exports.

> **The version installed in your org can be read in Setup → Installed Packages.** Object, field and custom metadata type API names are given here without the `orgguardian__` namespace prefix that they carry in your org (`Finding__c` is `orgguardian__Finding__c` there).

1. What runs on its own: the twelve scheduled jobs

A complete arming places **twelve scheduled jobs** in the org: an hourly scan, and **eleven** maintenance jobs. **Setup → Scheduled Jobs** lists them under the names below.

Job name (as it appears in Setup)	Frequency	What it does
`OrgGuardian Hourly Scan`	hourly	runs the scan described below
`OrgGuardian Data Retention`	nightly	applies the retention windows (§9)
`OrgGuardian Compliance Snapshot`	weekly	records the compliance evidence snapshot (DORA / NIS2 pack)
`OrgGuardian Risk Resurface`	nightly	brings expired accepted risks and lapsed acknowledgements back to the surface (§3)
`OrgGuardian SLA Compliance`	weekly	computes the SLA compliance report
`OrgGuardian Escalation Chains`	hourly	runs the escalation chains
`OrgGuardian Endpoint Health Check`	hourly	checks the health of the monitored endpoints
`OrgGuardian Board Report`	weekly	produces the board report and archives it
`OrgGuardian Fleet Sync`	every 4 hours	synchronises the linked orgs of the fleet
`OrgGuardian Posture Digest`	weekly by default, or daily: the cadence chosen in the **Posture digest** section	sends the posture digest

Job name (as it appears in Setup)	Frequency	What it does
`OrgGuardian ITSM Sync`	hourly	synchronises the status of ITSM tickets
`OrgGuardian Slack Reactions`	hourly	reads the reactions on Slack alerts

A job whose feature is off, or not included in the org's plan, runs and does nothing. This applies in particular to the four jobs that belong to outbound features, all off at installation: `OrgGuardian Endpoint Health Check` ("Endpoint health check" feature), `OrgGuardian Fleet Sync` ("Multi-org aggregation"), `OrgGuardian ITSM Sync` ("ITSM integration") and `OrgGuardian Slack Reactions` ("Slack reaction acknowledgement"). They are placed like the others and stay inert as long as their feature is off or outside the org's plan.

`OrgGuardian Slack Reactions` reads the reactions on Slack alerts ("Slack reaction acknowledgement" feature, `Chat_Reactions`, off by default, paid plan, `ENTERPRISE`; see "Acknowledging from Slack with a reaction", §5). It queues nothing until all its conditions are met — the `Chat_Reactions` and `Channel_Slack` features on, with the paid plan, the consent, the channel ID and the name of the bot's Named Credential.

Four of these jobs defer what one transaction cannot cover within the platform's 120-second window of callout time. `OrgGuardian Endpoint Health Check` splits its probes into chained slices of at most 9 endpoints, each in its own Apex job. `OrgGuardian Fleet Sync`: a fleet larger than what one transaction pulls continues in the Apex job `MultiOrgSyncBatch` (visible in Apex Jobs), in slices of 10 orgs, one at a time. `OrgGuardian ITSM Sync`: each ticket read waits at most 10 seconds; when the ITSM instance is slow, the pass stops before the window and defers the remaining tickets to the next pass (`ITSM_SYNC` log, `deferred|…` line). While a pass is partial, the sync findings do not close: they wait for a pass that has read its whole batch of tickets. `OrgGuardian Slack Reactions` leaves to the next pass the messages that the budget no longer covers.

`OrgGuardian ITSM Sync` also queues an **automatic ticket creation** pass (Apex job of the class `ItsmAutoCreateRun`), in its own transaction and never in a scan's, when the **Automatic ITSM tickets** feature (`Itsm_Auto_Create`, off by default, paid plan, `ENTERPRISE`) is on in addition to the ITSM integration, and never while a creation pass is already waiting or running (the log then says so). It creates nothing without a consent **separate** from the ITSM consent, given in the **Secrets and consents** pane of the **Settings** tab with a minimum severity (Critical, High or Medium). It opens at most **10 tickets per pass**, one per open finding at or above that severity, with no snooze or accepted risk in progress, other than the ITSM sync's own failure findings, that has **no tracking line** that blocks it (an open ticket, a line with no reference, or a refusal line less than **7 days** old), and that became eligible in the **last 24 hours**, since that consent: first seen, out of a snooze or an accepted risk that ended, reopened after a verified closure less than 24 hours old, or released by the expiry of its refusal line. The rest waits for the next pass, the most severe first. Lowering the severity moves the consent date to the moment you save (a finding still waiting for its ticket at the former severity loses it); raising it keeps it. Changing the destination or the tool withdraws the automatic consent, to be given again. Turning the `Itsm_Auto_Create` feature off suspends the policy without withdrawing it: when it is turned back on, only

the findings that became eligible in the last 24 hours get their ticket, 10 per pass. A ticket closed in your tool acknowledges its finding with a snooze; a finding still open when that snooze ends can get a new ticket, within the same bounds.

A payload that the personal-data guard refuses, or that your tool refuses as too large (HTTP 413), creates nothing: OrgGuardian records a **Refused** line for that finding, does not retry it automatically until that line expires (7 days; the finding is then a candidate again: one try per expiry, except a held HTTP 400 that nothing settles in its pass, see below), and the pass goes on. An HTTP 400 leaves a **Refused** line when another ticket of the same pass is accepted by your tool, before or after it: the configuration is then proven. An HTTP 400 received before any ticket of the pass is held and the pass goes on; a second HTTP 400 with no accepted ticket in between stops the pass and leaves a **Refused** line for each of the two (two sendings at most), because it most often means a configuration to review — the issue type "Task" missing from the Jira project (OrgGuardian creates its Jira tickets with that type), a field the project requires that OrgGuardian does not fill, an integration account that cannot create issues — and the reason says so, in the log and on the screen. The other findings wait for the next pass; those two are candidates again only when their lines expire. On a wrong configuration, where your tool refuses every sending with an HTTP 400, each pass records at most two **Refused** lines because of it: for 7 days, neither the pass nor a click creates a ticket for those findings, unless an administrator lifts the refusal (**Allow a new ticket**, below); at expiry, each finding whose refusal was not lifted gets a new automatic try. A held HTTP 400 that nothing settles before the end of its pass (no accepted ticket, no second HTTP 400) gets no line: its finding is retried at the next pass while it stays eligible, without blocking the others. An HTTP 422 (your tool reports a configuration problem), any other refusal of the tool (401, 403, 404, 429…), a transport failure before sending or a sending OrgGuardian itself refused stops the pass, which tries again the next hour. When OrgGuardian cannot know whether the tool created the ticket (a 5xx response, a timeout, a connection cut after sending), it records a line with **no reference**, stops the pass and never creates that finding again automatically: check the ticket in your tool. A ticket's description is cut to the tool's limits (32,767 characters for Jira, 4,000 by default for ServiceNow), with a note saying so. If a ticket that left cannot be recorded in Salesforce (full storage, a validation rule or a trigger on **ITSM Ticket**, a removed permission), the pass trips a **circuit breaker**: no automatic creation until an administrator re-arms it in the same pane, which says since when, how many tickets may have no line and the cause; an unexpected error after a sending trips it too, after recording the lines already obtained. Two exceptions: a tripping of the breaker that the platform refuses (a locked row, the setup data limit), written as an error in the log, after which the next pass may create those tickets again; and an error the platform does not let OrgGuardian catch — a limit exceeded, for instance by a trigger on **ITSM Ticket** — which also undoes the breaker and the log: if it happens again, each pass may create those tickets again, 10 at most per hour. The user who scheduled the job must be able to create **ITSM Ticket** records (`Itsm_Ticket__c`) and hold "Customize Application" (to trip the breaker): otherwise the pass stops before reading anything, sending nothing.

`ITSM_AUTO` log: every pass that reads its candidates writes its summary there (candidate findings, tickets created, refusals, failures, unknown outcomes, deferrals); a pass stopped before reading anything writes its reason there (permission to create **ITSM Ticket** records or to trip the breaker, signing key missing, breaker tripped, ITSM integration off) as soon as the automatic policy is consented and the feature is on, with no name and no value; nothing is written when the policy is not set up, the feature is off or the plan no longer covers it. A creation pass still queued or running at the next hour is written there too, and no other one is queued. These lines are written with the permissions of the user who scheduled

the job: without create access on the log, they are not kept. Withdrawing the ITSM consent, changing the destination or clearing the ITSM family also withdraws the automatic policy: the next passes then write nothing.

The two **Create a ticket** buttons, on a finding and on an incident, refuse a duplicate too: a finding that already carries a tracking line that blocks it gets no second ticket, and the screen names the existing reference; a **Refused** line older than 7 days no longer blocks. When a click is refused because of a **Refused** line, an administrator (the "OrgGuardian: manage features" permission or "Customize Application") sees **Allow a new ticket** next to the button, on the finding's record page or in the incident; lifting also requires edit access to ITSM tickets. The button turns the finding's active refusal lines to **Closed**, kept as a trace with no reference, and the next click creates the ticket. Once a refusal is lifted, the automatic pass no longer retries that finding when the 7 days end: create its ticket by hand. The two buttons send nothing when your user cannot create the tracking line. When OrgGuardian cannot know whether your tool created the ticket of a click (a 5xx response, a timeout, a cut), the button says so without recording a line: check the ticket in your tool before clicking again; meanwhile, the automatic pass may open a ticket for that finding if it qualifies.

The job names are fixed: this is how a new arming recognises the jobs already in place and never creates a duplicate.

Trigger times are spread per org. Each job's schedule is derived from a stable hash of the org Id plus a value specific to that job: the minute is spread across the hour, and the hour of the nightly and weekly jobs across the 01:00–05:59 window. Subscriber orgs therefore do not all fire at the same moment, and the jobs of a single org are spread as well. A given org always recomputes the same schedule, so a new arming changes nothing.

> `OrgGuardian Posture Digest` is the **only** job whose schedule depends on one of your settings: the cadence chosen in **Administration ▸ Settings ▸ Posture digest** (stored in `Digest_Config__mdt.Cadence__c`). A new cadence takes effect at the next opening of the OrgGuardian **Home** screen by a user who can write findings and metrics — until then, the section states that the chosen frequency is not yet applied — or at once, by running `orgguardian.OrgGuardianAdmin.scheduleMonitoringJobs()` in anonymous Apex, with the rights described in the Installation Guide ("Forcing the bootstrap if necessary"). A **daily** report by recipient (`Report_Template__mdt.Cadence__c`, **Reports by recipient** feature) also makes it run every night, applied the same way; the weekly digest and the remediation reminders then keep their day, Monday (§5).

What a scan does

A scan runs **six packs**, each in its own transaction, in this order (the names are those recorded in the **Packs Run** field, `Packs_Run__c`, of each scan run record, `Scan_Run__c`):

`Limits` · `Monitoring` · `Security` · `CalloutPosture` · `CodeQuality` · `Intelligence`

A pack that the org's plan does not include is skipped. On the `FREE` plan, the `Security` pack runs its free base only: five families of checks — human accounts without MFA, guest user exposure, public file links, the OAuth connected-app inventory, and the holders of Modify All Data and View All Data; its other

checks require a paid plan. Running each pack in its own transaction isolates failures: a governor limit hit in one pack does not cost the findings of the others. On a paid plan, the `Security` pack is prepared by two transactions of its own, which collect data exposure and then configuration; if one of them fails, the pack closes no finding and its score is not published for that scan. The scan then finishes in two further stages — closing the scan run record and computing the posture score on one side, the predictions and insights on the other.

The scan has no other scheduled entry point. The scheduled scan runs as the user who armed the jobs, the **scan account**. Besides writing findings and metrics, that account needs four read permissions that the package cannot grant: View Setup and Configuration, View Roles and Role Hierarchy, View All Users and View All Profiles (the Installation Guide §5.1 bis). Without them, the Security checks that depend on them are not evaluated.

The windows

- **`Scan_Window_Policy__mdt`** (Administration ▸ Settings ▸ **Detection window**) decides when a scan is allowed to run (`Start_Hour__c`, `End_Hour__c`, `Business_Hours_Only__c`) and whether it runs as a dry run (`Dry_Run__c`).
- **`Maintenance_Window__mdt`** (Administration ▸ Settings ▸ **Maintenance windows**) decides when alerts are silenced (`Days_Of_Week__c`, `Start_Time__c`, `End_Time__c`, `Affected_Scopes__c`, `Suppress_Alerts__c`).

---

2. Running a scan manually

On the **Home** screen, the **Setup tour** checklist offers **Run scan now** (step "Run your first scan"); the **Guided tour** offers the same action as **Run a scan now**, while nothing has been scanned yet. Both are offered only until the first scan completes. After that, scans run every hour, and a scan on demand is started in anonymous Apex with `orgguardian.OrgGuardianAdmin.runScanNow(false)` (the Installation Guide §5.3, which gives its own checks). Before any collector starts, the two buttons have the server apply **two checks**, in this order:

1. **Can what the scan produces be written?** The current user must be able to create **and** update `Finding__c` **and** `Org_Metric_Current__c`, the two objects that a scan writes. Otherwise, immediate refusal:

   > "You do not hold the rights required to launch a scan: creating and updating findings and metrics is required. Contact your administrator."

2. **Is a scan already in flight?** Two Lightning tabs open at the same time cannot queue two scans. Refusal:

   > "An OrgGuardian scan is already running or queued. Try again once the in-flight scan has finished."

Launching a scan therefore requires the **ability to write the results**, not **configuration authority**. Of the shipped permission sets, only `OrgGuardian_Admin` grants these rights: a reader (`OrgGuardian_Viewer`), an analyst or an auditor is refused before the first collector starts, not after.

---

3. Reading findings

The object and its states

A finding is a `Finding__c` record.

- `Severity__c` — five values: `Info`, `Low`, `Medium`, `High`, `Critical`.
- `Status__c` — three values: `Open`, `Resolved`, `Ignored`.
- Every action leaves a line in `Finding_Activity__c`, a register to which the product only ever **adds** lines; only the retention purge (section 9) deletes from it. For that purpose the `OrgGuardian_Admin` permission set carries `Delete` on the object, and therefore `Edit`, which the platform requires alongside `Delete`; `OrgGuardian_Analyst` has only `Create` and `Read` on it, `OrgGuardian_Auditor` and `OrgGuardian_Viewer` only `Read`. An acknowledgement set by a Slack reaction leaves one too, **with no actor**: see "Acknowledging from Slack with a reaction".

Lifecycle actions

On the **Findings** screen (its **Findings** view), each finding carries a **Finding actions** menu, and a selection of findings carries a **Bulk actions** menu. Both appear only to users who can edit findings — among the shipped permission sets, `OrgGuardian_Analyst` and `OrgGuardian_Admin`.

Action (as labelled on screen)	What it does
Assign to me / **Unassign**	assigns the finding to you, or removes its assignee
Snooze…	postpones the finding for the number of days you enter, with an optional reason
Acknowledge…	acknowledges the finding (you take charge of it), with an optional reason
Accept risk…	records an accepted risk for the number of days you enter, with a risk justification; you are recorded as its signatory
Create ticket…	opens a ticket in your ITSM tool (ITSM integration, off by default)
Bulk actions	applies **Assign to me**, **Snooze…**, **Acknowledge…** or **Accept risk…** to the selected findings

Apart from the ITSM ticket, these actions only annotate OrgGuardian's own finding records; none of them changes your org. The record page of a finding offers three of them as buttons: **Assign to me**, **Acknowledge** and **Create a ticket**.

An accepted risk does not disappear for good: the `OrgGuardian Risk Resurface` job brings it back to the surface when the period ends. Risk can also be accepted by rule rather than finding by finding: each rule of the **Risk accepted** pane (Administration ▸ Settings), stored in `Acceptance_Policy__mdt`, accepts

the findings whose deduplication key (`Dedupe_Key__c`) matches its pattern — optionally limited to a category and a minimum severity — with a reason, an approver and an expiry date (paid plan).

An acknowledgement is time-boxed as well: it lasts `Ack_Resurface_Days` days (general settings register, 30 by default), after which a finding that is still open returns to the queue. The same nightly job applies this delay to acknowledgements that have outlived it, oldest first and at most `Ack_Resurface_Max_Per_Run` per pass (25 by default; 0 disables it).

The screens

One tab is dedicated to findings, **Findings** (`OrgGuardian_Tab_Findings`): its **Findings** view is the list of findings, and every link that opens the list (Home tiles, the search palette, Ask the org's **Open in Findings**, the Security screen's chips) opens it in this tab, filtered on what the link counts. Each finding also has its own record page.

---

4. Dashboards and reports

What the package ships

In your org, the reports and dashboards below sit in the **OrgGuardian** folder of the Reports tab and in the **OrgGuardian** folder of the Dashboards tab.

- **Three dashboards**:
  - `OrgGuardian_Posture_Overview` (OrgGuardian Posture Overview), which assembles the four findings and score reports;
  - `OrgGuardian_Compliance_Overview` (OrgGuardian Compliance): attestations by framework and open findings by category;
  - `OrgGuardian_Operations_Overview` (OrgGuardian Operations): endpoints by status and incidents by peak severity.
- **Nine reports**: `OrgGuardian_Org_Health_Score` (Org Health Score), `OrgGuardian_Open_Findings_By_Severity` (Open Findings by Severity), `OrgGuardian_Open_Findings_By_Category` (Open Findings by Category), `OrgGuardian_Findings_Trend` (Findings Trend (by Month First Seen)), `OrgGuardian_Attestations_By_Framework` (Attestations by Framework), `OrgGuardian_Endpoints_By_Status` (Endpoints by Status), `OrgGuardian_Incidents_By_Severity` (Incidents by Peak Severity), `OrgGuardian_Insights_By_Type` (Insights by Type) and `OrgGuardian_Fleet_Health` (Fleet Health by Org). The last two appear on no dashboard.
- **Seven custom report types**, so that the customer can build their own, each based on a package object: `OrgGuardian_Findings` (`Finding__c`), `OrgGuardian_Metrics` (`Org_Metric_Current__c`), `OrgGuardian_Attestations` (`Attestation__c`), `OrgGuardian_Endpoints` (`Endpoint_Status__c`), `OrgGuardian_Incidents` (`Incident__c`), `OrgGuardian_Insights` (`Insight__c`) and `OrgGuardian_Fleet` (`Linked_Org__c`).

The product screens

Most of the reading happens in the **OrgGuardian** app, whose navigation bar carries eight tabs: **Home**, **Executive**, **Findings**, **Monitoring**, **Connectivity**, **Alerts**, **Drift & audit** and **Administration**. The package ships 43 tabs in all (see the Configuration Guide). The main tracking screens: **Dashboard**, **Monitoring**, **Trends**, **Charts**, **Limits**, **Security**, **Compliance**, **Insights**, **Executive summary**, **Executive**, **Risk heat map**, **Config impact**, **Fleet**, **Incidents**, **Weekly health grid**.

What each figure of the Home page says. The large figure is the org **health score**: the share, weighted by severity, of the controls evaluated by the last scan that passed, over every control of the catalogue (logins and org limits included), with its word under the ring (Healthy, Needs attention, At risk, Partial or Not measured). The pills of the "Pack scores" row (Security, Code quality, Monitoring) are **pack scores**: the same rate, over the pack's controls only (the Configuration Guide, "How pack scores are computed" and "How the health score is computed"). The health score is not the average of the pills: it counts each evaluated control at its weight, those that no pack scores included.

A custom rule that targets a pack score or the health score compares its threshold with a share of passed controls. A rule (`Detection_Rule__mdt`) whose metric key designates `Security:Score`, `CodeQuality:PostureScore`, `Callouts:PostureScore`, `Monitoring:PostureScore` or `Org:HealthScore` reads that share, not a count of findings. The value can also be empty ("Not measured"): a rule never fires on an empty value. Set the threshold and the operator of these rules accordingly.

Trend data lives in the seven Big Objects — `Org_Limit_Snapshot__b`, `Config_Snapshot__b`, `Compliance_Snapshot__b`, `Em_Metric_Snapshot__b`, `Login_Baseline__b`, `Prediction_Outcome__b`, `Endpoint_Health_Snapshot__b` — and the current state in `Org_Metric_Current__c`.

---

5. Alerts and outbound channels

The defining fact: no outbound call by default

No external alert channel can send on a default installation. This is not an operating recommendation; it is the product's default behaviour. More broadly, the package's **22 outbound features**, in four families — alert channels (5, reading Slack reactions included), exports and integrations (4), external probes (7) and calls to your own org's API (6) — are all off at installation (see the Security FAQ, §3).

Beyond the two default channels internal to the org (described below), five alert channels each sit behind **their own** feature, off by default:

Channel	Feature (label in Feature activation)	Network egress
Slack	Slack alert channel (`Channel_Slack`)	yes
Microsoft Teams	Teams alert channel (`Channel_Teams`)	yes
PagerDuty	PagerDuty alert channel (`Channel_PagerDuty`)	yes
Webhook	Webhook alert channel (`Channel_Webhook`)	yes
Chatter	Chatter alert channel (`Channel_Chatter`)	no: it posts inside your org

A feature is available only when its switch is on **and** the org's plan includes it, and all five channels require a paid plan. Without a paid plan provisioned by the publisher, an org runs the **30-day trial of the full plan** from the first screen opening by an administrator, then falls back to the `FREE` plan, which includes none of the five. The plan alone opens nothing: with the five switches off — the default posture, during the trial as after it — **none of the five channels can send**.

The double lock: plan and administrator consent

- **Lock 1 — the plan and the feature switch**: the channel's feature is switched on in **Administration ▸ Settings ▸ Feature activation**, and the org's plan includes it.
- **Lock 2 — the administrator's consent**: a destination must have been registered explicitly, on the **Alert channels** screen, in the **protected** custom setting `OrgGuardian_Channel_Settings__c`; the screen accepts it only once lock 1 is open. For the four outbound channels, this destination is a **Named Credential DeveloperName**, never a plain-text URL (`Slack_Named_Credential__c`, `Teams_Named_Credential__c`, `PagerDuty_Named_Credential__c`, `Webhook_Named_Credential__c`); the fifth, Chatter, stores a Chatter group identifier there (`Chatter_Group_Id__c`), hence no network egress. Teams stores nothing else there; for its reactions, Slack adds the name of the bot's Named Credential, the optional list of allowed members, the channel ID `Slack_Channel_Id__c` and the consent instant `Chat_Reactions_Consent__c`, never a secret (see "Acknowledging from Slack with a reaction"). Two channels add a **secret**, entered by your administrator in **Administration ▸ Settings ▸ Secrets and consents**: the PagerDuty routing key (`PagerDuty_Routing_Key__c`) and the webhook's HMAC signing secret (`Webhook_HMAC_Secret__c`). The webhook also requires an explicit **consent**: the box **Allow alerts to be posted to the webhook** (`Webhook_Consent_Granted__c`, dated by `Webhook_Consent_At__c`, with no identity), on the **Webhook signature and consent** card of the same screen as its secret, which refuses it while the Named Credential or the secret is missing. The consent falls when the box is unticked, when the card is cleared (**Clear**: the secret and the consent go together), or when the Named Credential is emptied on the **Alert channels** screen. That screen accepts only a Named Credential **name** for the four outbound channels, and refuses, for the webhook, one whose URL it reads as `http://`. Each of the four channels checks the stored value again before sending: a value that is not a name (a path, a query or a URL) configures nothing, the screen and the setup step show the channel as not configured, and nothing is sent; for Slack, Teams and PagerDuty, the channel's log names the cause (`NC_SHAPE`). Without a Named Credential name, a secret and a consent, nothing is sent: the webhook signs and timestamps every send. What the receiving service gets, and how it verifies it: the Webhook Receiver Guide (available on request). The setting carries no direct address for the webhook receiver: the webhook goes out only through its Named Credential.

The two locks are **cumulative**: an open feature with no configured destination sends nothing, and neither does a configured destination with no open feature. The **Alert channels** screen shows, for each channel, which lock is closed — that is the diagnosis to read before looking anywhere else:

- "Not enabled": lock 1 is closed (feature off, or not included in the plan);
- "Included, Named Credential missing" ("Included, Chatter group missing" for Chatter): lock 2 is closed;

- for the webhook only: "Included, signing secret missing" or "Included, consent missing". Its Named Credential is registered, but the signing secret, or the consent, of **Secrets and consents** is not: nothing is sent until both are;
- "Ready (enabled and configured)": both locks are open. Only a ready channel offers **Send a test alert**. If the answer is "Test not delivered", the refusal is downstream (outbound guard, endpoint or target response): the `OUTBOUND_BLOCKED` entries of the system log (`System_Log__c`) give the reason of a refusal before sending; for the webhook, the `WEBHOOK_DELIVERY` entries give the receiver's answer: one per attempt, including an attempt that Salesforce refuses at sending (`status:transport`).

The same pattern applies to the **Heroku option**: publishing requires the option to be available — its switch on and a paid plan, or a one-time 30-day trial of the option, started by an administrator and not muted — **and** the consent recorded in the protected setting `OrgGuardian_Heroku__c` (`Consent_Granted__c`) **and** a configured endpoint **and** the HMAC signing key (`Hmac_Key__c`, same protected setting; Troubleshooting Guide §11). Consent is never cached: once revoked, the revocation takes effect at the next scan. When the consent cannot be read, the product treats it as not given.

Acknowledging from Slack with a reaction

A reaction added to an alert's message, in the configured Slack channel, acts on that alert's finding: the green check mark (`:white_check_mark:`) **acknowledges** it — escalation stops for the acknowledgement window (30 days by default, adjustable from 1 to 365 days: the setting that also bounds an acknowledgement made on screen), then resumes if the finding is still open; the immediate alert a scan would send again for that finding is suspended for 24 hours; but the finding stays in the score, each end-of-scan summary, the SIEM export and the automatic ITSM ticket creation: no snooze is set. **Only the immediate alert of a Critical finding carries the line that invites a reaction**: the green check mark therefore acknowledges the finding of a Critical alert. "zzz" (`:zzz:`) would snooze a Medium, Low or Info finding for 24 hours, but no alert of those severities carries the line: that snooze has no message to act on, and on a Critical finding "zzz" sets nothing (the log says so). No other reaction does anything.

- **No call enters your org.** OrgGuardian receives no call from Slack: there is no public site, no entry point, no signature to verify. The org itself **reads**, every hour, the reactions on the messages it posted (`OrgGuardian Slack Reactions` job). A reaction is therefore taken into account within the hour as long as at most 25 messages are watched; beyond that, passes rotate (see "Bounds").
- **A bot, through its own Named Credential.** An incoming webhook does not return the identifier of the posted message: to read its reactions, the channel then posts as a Slack bot, through the Web API (`chat.postMessage`), then reads the reactions (`reactions.get`). Slack scopes to grant the bot: `chat:write` and `reactions:read`, nothing else; invite the bot to the channel. The Named Credential of your **incoming webhook**, on the **Alert channels** screen, does not change. A consented bot is enough, even with no incoming webhook: that screen then shows Slack as "Ready (enabled and configured)", the test alert leaves through the bot, and the "Configure an alert channel" setup step is done.
- **The bot token never goes through OrgGuardian's code.** Create a Named Credential (Setup → Named Credentials) to `https://slack.com`, with an External Credential (External Credentials tab) that carries the header `Authorization: Bearer <bot token>`; allow the package namespace, `orgguardian`, among the namespaces allowed to call out from managed packages (the Named Credential's `AllowedManagedPackageNamespaces` setting; without it, the platform refuses the

package's call). Give access to its principal (permission set → External Credential Principal Access) to every user under whose identity a Slack alert leaves: the user who scheduled OrgGuardian's jobs (scans, escalations, the hourly reading of reactions) and anyone who runs a scan by hand, a drill, the test alert or a fleet pull ("Pull from linked orgs", **Fleet** screen). Enter the **name** of this Named Credential on the "Slack reactions" card, **Slack bot Named Credential** field (a name, no path). No OrgGuardian field, log or screen holds the token.

- **Four locks, plus the channel.** The "Slack reaction acknowledgement" feature (`Chat_Reactions`, off by default, paid plan, `ENTERPRISE`, never opened by "Enable everything possible"); the Slack channel itself (`Channel_Slack`); and, on the "Secrets and consents" screen, "Slack reactions" card, the **channel ID** (it starts with C or G and is shown in the channel details), the name of the bot's Named Credential and the **consent** box, which refuses as long as that name is missing. No message posted before the consent in force is read. The user who scheduled the jobs must be able to edit the findings — the message reference (`Chat_Message_Ref__c`, which only the `OrgGuardian_Admin` permission set writes) and the acknowledgement and snooze fields — and to create their `Finding_Activity__c` lines: otherwise the pass reads nothing, and the `CHAT_REACTIONS` log says why. If the identity a scan runs as cannot read the acknowledgement time or origin (`Acknowledged_At__c`, `Ack_Source__c`), the scan still sends its summary, but does not suspend the immediate alert of a finding acknowledged by a reaction. Escalation, which runs as the same scheduled identity, reads both fields with no fallback: without that read, it reads no finding and escalates nothing, and only an `ESCALATION_CHAINS` log line says so, on every pass. Saving the "Slack reactions" card requires the "Customize Application" permission: without it, the save is refused, the screen says so and nothing is written.
- **Withdrawing, with no further step.** Unticking the box and saving, or clearing the card, withdraws the consent and forgets the channel; turning the `Chat_Reactions` feature off has the same effect on sending. In these cases, alerts go back to the incoming-webhook format, through **its** Named Credential, which nothing has touched. A plan that falls below `ENTERPRISE` closes the Slack channel itself (`Channel_Slack` is `ENTERPRISE` too): no Slack alert leaves, by either route, and the card says so. If you never set an incoming webhook, Slack alerts no longer leave: the card warns you, and the `SLACK_ALERT` log names it on every alert. An incoming webhook is counted as delivered only on its `ok` response: any other response (an HTML page, another address) is a non-delivery, named in the same log.
- **Who counts, and what remains.** A Slack member is never matched to a Salesforce user: the acknowledgement is recorded **with no author** (`Acknowledged_By__c` empty), with origin `CHAT` (`Ack_Source__c`, shown on the finding's page), and each effect leaves a `Finding_Activity__c` line **with no actor**, which names the reaction and the message. Who reacted is shown on the Slack message that `Chat_Message_Ref__c` points to (readable by the `OrgGuardian_Admin` and `OrgGuardian_Auditor` permission sets), as long as the reaction stays there: removing it undoes nothing and erases that name. No email address is requested from Slack. On the card, an **optional** list of allowed members (Slack member IDs, `U…` or `W…`, separated by commas) restricts the reactions that count: when set, only theirs act; the identifiers Slack returns with the reactions are compared to that list in memory, then forgotten — neither kept, nor written, nor logged. **Without a list**, anyone — or any app — who can react in that channel (a guest, or in a Slack Connect shared channel a member of another organisation) can acknowledge the finding of an alert: set the list, or keep the channel for your on-call team (private channel, not shared). The

Change feed screen, which lists the Setup Audit Trail findings, names the latest Salesforce actor of each: after a reaction, it still shows the user who acted before it. The activity line with no actor, in the finding's **Activity** on the **Findings** screen, is what names the reaction.

- **What a reaction never does.** It does not resolve a finding (the scan closes it once the problem is gone); it acts only once per message (a reaction left on the message is not read again); it never takes precedence over a decision made during the reading (an acknowledgement, a snooze or an accepted risk set during the pass prevails), nor over an acknowledgement or an accepted risk set in Salesforce after the alert: both date the acknowledgement (`Acknowledged_At__c`), and the message is no longer read; it acts only on the **latest** message that carries the line inviting a reaction for that finding — an escalation step does not carry it and does not replace that message, a new alert of the finding replaces it —; it acts only on the alert of a recorded finding: never on a digest, a grouped burst, a report, a score drop, a fleet or endpoint alert, a drill or a test send — these messages do not carry the line. Two exceptions to know. A snooze set on screen after the alert does not date the acknowledgement: the message is not read while it runs, but if it ends less than 72 hours after the alert, a green check mark left on the message acknowledges the finding at the next pass. An accepted risk keeps its acknowledgement date until the nightly `OrgGuardian Risk Resurface` job clears the expired acceptance: if that run falls less than 72 hours after the alert, the finding becomes readable again and a green check mark left on the message acknowledges it at the next pass. Remove the reaction if that is not intended.
- **Bounds.** At most 25 messages read per pass, for 72 hours after they were sent; beyond 25, passes rotate: each message is read again about every N hours, N being the number of watched messages divided by 25 and rounded up. A pass keeps only the 500 oldest watched messages: only they are read again within 20 hours at most; beyond 500, the most recent ones wait until older ones leave the 72-hour window or are handled. Slack's first refusal (token refused, channel not found, rate limit) stops the pass, which tries again the next hour. Each pass that reads messages writes to the `CHAT_REACTIONS` log one line per finding acknowledged or set aside (the finding and the message, never a person) and its summary; a pass closed by a configuration lock writes nothing there, a pass closed for lack of permissions says so.
- **Microsoft Teams is not covered**: reading the reactions on a Teams message requires Microsoft Graph and a Microsoft 365 tenant administrator consent. Teams keeps the "Acknowledge in Salesforce" link.

What sends by default: two channels internal to the org

In the absence of any `Alert_Channel_Config__mdt` record — and the package **ships none** (**Setup → Custom Metadata Types → Alert Channel Configuration → Manage Records** is empty after installation) — alerts go to two channels internal to the org:

- **A platform event**, `Org_Health_Alert__e`. The default payload carries only **coordinates** (source, severity, category, finding identifier): the advisory text is behind the **Finding detail in Platform Event alerts** feature (`Alert_Event_Detail`), off by default. A subscriber who wants the detail reads `Finding__c` itself, under its own access rights.
- **A custom notification** of the shipped type **OrgGuardian Alert** (`OrgGuardian_Alert`), declared for desktop only (the Installation Guide §2.2): the bell stays in the org. Its title is `[<severity>] <alert type>`, at most 250 characters, and its body at most 750 characters, the platform's limits for a custom notification. On the scheduled analysis, the default path, it is the end-of-scan digest: title `[Critical] BATCH_DIGEST`, whatever the severity of the findings summarized; body "OrgGuardian

scan summary: ...", listing up to ten findings as `[<severity>][<category>] <source> -> <target>` (a target can be a user Id, a host or an endpoint URL), cut at 750 characters. An immediate alert (a Critical under escalation) carries the finding's sentence as the screens render it, within the same limit. An escalation step carries that sentence behind its header. The notification targets the finding (the first one of the digest): opening the notification displays it. A notification with no saved finding (posture digest, board report, SLA digest, incident drill, endpoint transition, Health Check score drop, org health score drop, fleet degradation) is sent with the reference of the OrgGuardian tab it concerns, the home page for the incident drill and for the org health score drop. Measured on a development org on 2026-10-03: the bell is sent with the reference of each of these tabs, and also towards a tab that does not exist, the platform not checking the target at send time; that a click on the bell opens the screen is not measured. The remediation reminders (below) use the same type.

The **email** channel is not a default: it requires an active `Alert_Channel_Config__mdt` record of type `EMAIL`, created by the administrator in **Administration ▸ Settings ▸ Channels and templates**, and a paid plan (the 30-day trial of the full plan counts). Its default recipients are the holders of the permission set `OrgGuardian_Admin`, and its template lives in `Email_Alert_Template__mdt`.

The alert sender address

Without any setting, an email alert is sent under the identity of the user who runs the send — for scheduled scans, the user who scheduled them: the message carries that person's email address, under the display name of the template, and replies go back to that person, unless the template (`Email_Alert_Template__mdt`) sets a reply-to address (`Reply_To__c`), which then receives the replies in every case. The administrator can pick an **organization-wide address** to send the alerts from: it changes the displayed sender (*From*) only; the scheduled scan and its send still run under the identity of the user who scheduled them. The address is set on the **Alert sender address** card at the top of the **Alert channels** screen (the **Alert channels** pill of the **Alerts** space). The address is stored in the protected setting `OrgGuardian_Channel_Settings__c` (`Email_Sender_Address__c`); no other screen writes it.

Preparing the address, in **Setup → Organization-Wide Addresses**:

1. add the address, give it a purpose that includes user selection (*User Selection*), and allow it for **all profiles**;
2. follow the link in the verification email that Salesforce sends to that address: until then, the address exists but is not verified;
3. enter it on the card, then **Save**.

What the card shows, and what the send does:

The card says	The send
"Not set"	goes out from the user who scheduled the scan (the default)
"sent from this verified organization-wide address"	goes out from that address
the address is not among the organization-wide addresses	goes out from the user who scheduled the scan

The card says	The send
the address is not verified yet	same: Salesforce refuses to send from an unverified address (measured on 2026-10-06), so using it would make the whole send fail
the address is set as the default no-reply address only	same: nothing says that Salesforce accepts it as the sender of a message, and the alert does not take that risk
the address is not allowed for all profiles	same: using it would make the whole send fail
the organization-wide addresses could not be read	the address is checked again before every alert; same if it cannot be used

The alert is therefore always sent: when the address cannot be used, the send falls back to the user who scheduled the scan, and the `ALERT_SENDER` log gives the reason. "Verified" is read from the `IsVerified` field of the organization-wide address, which the Salesforce documentation describes from API version 58.0 (*Object Reference*, `OrgWideEmailAddress` object, Summer '26 edition, read on 2026-10-06); the purpose, from its `Purpose` field, which the same page describes as follows: “UserSelection allows users with the correct profile to select the address as the From address for an email”. It says nothing of a `DefaultNoreply`-only address set as the sender of a message sent by Apex: OrgGuardian does not take that risk.

When saving:

- an address whose failure is **proven** — not among the organization-wide addresses, not verified yet, set as the default no-reply address only, or not allowed for all profiles — is **refused**, the message says what to fix, and nothing is changed;
- when the organization-wide addresses cannot be read, the address is accepted, and the card says so instead of calling it usable;
- **Clear** removes the address: the alerts go out from the user who scheduled the scan again;
- saving requires the OrgGuardian administration authority (permission set `OrgGuardian_Admin`, or the "Customize Application" permission) and, the setting being protected, the "Customize Application" permission itself: without it, the save is refused, the message names that permission, and nothing is written. The write is read back: the screen never says "saved" for a write the platform ignored. The same read-back applies to the destination of each extended channel, on the same screen.

The **Alert channels** pill is reserved for administration: a reader sees "Reserved for OrgGuardian administrators" there. If checking your access fails while you hold the `OrgGuardian_Admin` permission set, the view shows "The channel settings could not be opened" and a **Retry** button, instead of telling you that you are not allowed.

Posture digest, remediation reminders and reports by recipient

The `OrgGuardian Posture Digest` job carries three sends, all at the job's night hour, and all off by default except the first:

1. **The posture digest** (`ADMIN` template), to the alert recipients. Its open-findings line also says how many of them are **regressions** ("including N regressions (resolved, then back)"): findings that came back after their closure, outside the families whose closure is not a fix (the Glossary §7); a refused read is said "not measured", never zero. With the `Remediation_Reminders` and `Pack_Dora` features on, it ends with the **REMEDIATION** block of the register (the Configuration Guide §3).
2. **The remediation reminders** (**Remediation reminders** feature, `Remediation_Reminders`, paid plan, with the DORA / NIS2 pack): one in-app bell per owner and per run, which covers that owner's overdue gaps, at most once a day per control, at the frequency of the digest (weekly by default). It rings only for an active internal owner who reads the register (the Permissions and Personas guide §3); when personal alert preferences (`Alert_Self_Service`) are on, the owner's own apply to it, and quiet hours that cover the night then hold it back at every run, which the log counts. An accepted risk is never overdue; a reopened gap starts again without the owner and the due date it had before its closure. The bell names no one; the `REMEDIATION_REMINDER` lines of the system log count the owners left out, by reason.
3. **The reports by recipient** (**Reports by recipient** feature, `Report_Routing`, paid plan): the executive and CISO reports whose template names a permission set are emailed to its active internal members, at their own frequency (`DAILY`; `WEEKLY`, on Mondays; `MONTHLY`, on the first Monday of the month), with only the blocks that the permission set itself reads; the executive template can also forward the weekly board report, its PDF without names. These emails are bound to their audience: neither personal preferences nor quiet hours hold them back. Each message carries its recipients in blind copy (Bcc), so they do not see each other's addresses. In an org that sets a Compliance BCC Email address, Salesforce refuses any Bcc: the same messages are then sent again with the recipients in "To", where they see each other. Each report sent leaves an `ALERT_DISPATCH` line in the system log (`System_Log__c`), without any address; the `PERSONA_REPORT` lines name a report not sent and its reason (empty or unreadable audience, no readable block, no active email channel), an external member left out or a withheld block.

By email, the digest and the reports go out in a **report** layout, not in that of a finding alert. The **Since the previous scan** card at the top of the **Dashboard** screen is not sent: it is read on screen. It states the moves of the health score without naming any category as their "main source": the score does not break down into penalties per category.

Reducing noise

Four mechanisms, all no-code. The last three are off by default. The first is on, but its cooldown period requires a paid plan: it is active during the 30-day trial of the full plan and on a paid plan, and stays off on the `FREE` plan. Only in-memory deduplication, within a single transaction, always applies.

Mechanism	Feature (label in Feature activation)	Default
Deduplication	Alert deduplication (`Alert_Deduplication`)	switch on, paid plan: **inactive on the `FREE` plan**
Grouping	Alert grouping (`Alert_Grouping`)	off, paid plan
Flap suppression	Alert flapping suppression (`Alert_Flap_Suppression`)	off, paid plan
Quiet hours (`Recipient_Delivery_Pref__mdt`)	Quiet hours (`Quiet_Hours`)	off, paid plan

The corresponding thresholds are in the general settings register (Administration ▸ Settings ▸ **General settings**): `Alert_Mass_Threshold` , `Alert_Cooldown_Minutes` , `Alert_Flap_Threshold` , `Alert_Flap_Window_Minutes` , `Alert_Correlation_Min_Pct` .

An end-of-scan alert goes out on a change, on every plan: a finding that is new or recreated (deleted, then detected again), reopened (closed, then back), or more severe than at its last alert. A finding that stays open and unchanged is alerted again only at its reminder, at the next scan: a Critical finding once `Alert_Reminder_Hours` have passed since its last alert (24 hours by default, 1 to 168), a High or Medium finding once a week. The end-of-scan summary and the immediate alert of a Critical finding each follow their own last delivery: an immediate alert held back by a maintenance window, or that no channel delivered, goes out at the next scan even when the summary already went out. An alert that is not sent again writes no log line. The `Alert_Cooldown_Minutes` window still applies on top of it. A summary that no channel delivers writes no delivery line: it goes out again at every scan, and every scan writes one non-delivery line per summarised finding (`ALERT_SUPPRESSED` , reason `CHANNEL_NO_EGRESS`) until a channel delivers it; the storage guard below drops them at its threshold.

Storage guard. When the org's data storage (`DataStorageMB`) reaches `Storage_Guard_Pct` (90% by default, 50 to 100), OrgGuardian stops its non-essential writes to `System_Log__c` (info and warning log lines, audit lines of alerts that were not sent) and says so in a Limits finding (`OrgLimit:StorageGuard`). Findings, current state, errors and the record of alerts sent are always written; writing resumes on its own below the threshold. Data storage can only be read by a user who can see it (a system administrator can; a user on a minimal profile holding `OrgGuardian_Admin` cannot): when the identity that runs the analysis cannot read it, the same finding says the guard is blind, and that the data storage limit is not measured either. The platform reports data storage in whole megabytes: the guard engages once the free space read is no more than the free share of the threshold rounded up to a whole megabyte, so 4 MB used of 5 on a Developer Edition at the default threshold. How the platform rounds a fractional consumption has not been measured. The guard cuts neither findings, nor current state, nor the `Scan_Run__c` line each scan writes (one an hour, kept `Retention_Log_Days` : 30 days by default, 7 on the free plan and 7 by default on a small org, section 9): in steady state, that is the largest part of the package's footprint. When a source of the Monitoring pack stays unreadable (`ApexUnexpectedException` ,

absent from some editions; `AsyncApexJob` refused to the identity that runs the scan), its warning line (`MONITORING_EXCEPTION_SOURCE_UNREADABLE`, `MONITORING_ASYNC_SOURCE_UNREADABLE`) is written at most once a day per source, not at every scan; an access refusal is still named at every scan by its finding.

Every send is traced by a `System_Log__c` line, including when the channel returned **without having sent** (lock closed, Named Credential missing, empty audience, caught exception) — the reason is a stable token, not a sentence, so that a monitoring tool can count the non-deliveries. On the `FREE` plan, only the lines of alerts actually sent are written: the non-delivery lines belong to the alert history (**Alert history** feature, `Alert_History`, switch on, paid plan).

---

6. Exports

Surface	Screen	Lock
Findings as CSV	**Findings** (its **Findings** view): **Export CSV**, which exports the list as filtered on screen	open to the `Analyst`, `Auditor`, `Admin` permission sets — **not** to `Viewer`
Configuration and posture as JSON / CSV	**Executive summary**, **Config & posture export** card: **Export config (JSON)**, **Export posture (JSON)**, **Export posture (CSV)**	**Config & posture export** feature (`Config_Export`), **off by default**, paid plan; the configuration export also requires the "OrgGuardian: manage features" custom permission, or the "Customize Application" or "Modify Metadata Through Metadata API Functions" privilege, while the posture export (findings, metrics, last scan) is open to the `Viewer`, `Analyst`, `Auditor` and `Admin` permission sets
Compliance evidence	**Compliance**: **Past reports**, the compliance posture as of a past date, the chain verification, the attestations and the **Remediation register**	**DORA / NIS2 pack** feature (`Pack_Dora`), switch on, paid plan; read access open to the `Viewer`, `Analyst`, `Auditor` and `Admin` permission sets, not to `OrgGuardian_Financials`, which carries only a custom permission; **Attest** and **Plan the remediation** require `Create` on `Attestation__c`, hence `Auditor` or `Admin`
Board report	**Executive summary**: **Export board pack** (self-contained HTML file); weekly job `OrgGuardian Board Report`	**Board report** feature (`Board_Report`), switch on by default but paid plan: available during the 30-day trial of the full plan, **invisible on the `FREE` plan**
Board report archives	**Executive summary**, **Board report archives** card: **Show archives** lists the archived board reports, newest first, with their date, reference (`BRD-…`), health score and DORA / NIS2 coverage; **Download** saves one as an HTML file, exactly as stored on its date, nothing recomputed, and checks its fingerprint	**writing** new archives follows the **Board report** feature (`Board_Report`, paid plan, available during the 30-day trial of the full plan); **reading** them does not depend on the plan: after a downgrade, the card stays on the **Executive summary** as long as an archive exists. The **purge** applies the configured window (`Retention_Posture_Report_Days`, 400 days by default) whatever the plan (section 9); the card states the window the purge applies (*The nightly retention purge deletes each archive … days after it was stored*). Read access open to the `Viewer`, `Analyst`, `Auditor` and `Admin` permission sets; an archive that prints amounts is served only to a reader allowed to see them (`Financials_Visibility_Mode` setting, `OS_View_Financials` custom permission)

The reader (`OrgGuardian_Viewer`) has no findings CSV export: they see the findings on screen. This is a deliberate separation between viewing and the extraction surface, with one exception: once `Config_Export` is on, the posture export of the **Executive summary** screen lists the findings, without their target for this set.

Integrity of the board report archives. When it is written, each archive is sealed with a SHA-256 fingerprint computed over its date, its two scores, its HTML body and the fingerprint of the previous archive (`Posture_Report__c.Archive_Hash__c`, `Prev_Hash__c`). Each download recomputes it and names the verdict in the message shown with the download:

Verdict	What it means
verified	the content matches its fingerprint, and the link to the previous archive holds (or the archive starts the chain)
modified	the content read back no longer gives the fingerprint written when it was archived
chain broken	the link to the previous archive does not match: an archive in between was deleted, or the previous one was modified and fingerprinted again, or stripped of its fingerprint while still kept. One case is not a tampering: when the user who schedules the board report has changed, the retention purge can be refused the older archives of the former owner (the board report archives are shared read-only, and the refusal is named in the purge's log) while it deletes the newer ones of the current owner; the first archive kept after them then reads **chain broken**
link cannot be checked	the content matches, but no earlier archive is kept any more (removed by the retention purge, which deletes the oldest first, or deleted): the link cannot be checked
no fingerprint	the archive carries no fingerprint, and nothing vouches for its content: the fingerprint was removed after archiving, the archive was inserted outside the scheduled board report, or the scheduled board report itself stored it without a fingerprint, because the fingerprint of the previous archive could not be read, or because the user who schedules it cannot create the two fingerprint fields (see "If you build your own permission sets" in the permissions guide). The message is a **warning**: the seal is part of the installed version, so no archive in your org can predate it
not checked	the fingerprint could not be read with the reader's permissions

The file is served whatever the verdict: it is your record, and the message says what the check found. What the fingerprint does not prove: it makes a modification **visible**, it does not prevent it. The `Admin` permission set keeps edit access to the object, because the retention purge deletes expired archives and the platform does not grant delete without edit; a holder of that set who recomputed the whole chain up to its latest archive would not be detected, since the fingerprint lives in the same org as the archive. Removing the fingerprint of an archive, on the other hand, shows: the archive reopens with **no fingerprint**, and the archive after it, if it already exists, with **chain broken**. Deleting the oldest archive kept cannot be told apart from the purge, nor can deleting the latest one before the next is written. The `Analyst` permission set has neither edit nor create access to this object: the scheduled board report, run under the `Admin` set, is the only writer.

---

7. What the product reads, and from where

The detailed reference is the publisher's **data source matrix**, available on request: it details, source by source and column by column, what is read, what is kept verbatim, what is hashed, what is reduced to a host and what is removed.

A single masking step handles Event Monitoring data: **every** Event Log File CSV line passes through it **before** anything is stored or sent. The model is an **allow-list per event type** — an undeclared column is removed, including a column that Salesforce might add in a future release — backed by a **hard refusal** that takes precedence over everything, covering `USER_ID`, `USER_ID_DERIVED`, `CLIENT_IP`, `SOURCE_IP`, `SESSION_KEY`, `LOGIN_KEY`, `URI`, `URL`, `USER_NAME`, `REQUEST_ID`.

---

8. What changes with Event Monitoring, and with Shield

Without Event Monitoring — the default case

The **Event Monitoring ELF fetch** feature (`EM_FETCH`) is **off by default** and requires a paid plan. On a default installation, **no Event Log File line is read**, and the masking step never runs.

The detectors that do not depend on Event Monitoring cover the essentials: org limits, health score, login failures, asynchronous performance, API versions, security posture, connectivity posture, code quality. Every detector of the `FREE` plan is among them.

With Event Monitoring

The product probes the org and distinguishes two paths:

- **The three streaming Event Log objects** — `LightningPageViewEventLog`, `ApexUnexpectedExcpEventLog`, `LightningErrorEventLog`. Their presence in the global describe means that the org can stream the events directly, without going through the CSV.
- **The paid `EventLogFile` event types** — `LightningPageView`, `LightningPerformance`, `ApexExecution` — whose presence attests to the full tier.

A missing `ViewEventLogFiles` permission degrades cleanly: the product treats Event Log Files as unavailable and carries on; nothing breaks.

When an administrator opens the tier, the enriched collectors become available: EPT p75, slow SOQL, unhandled exceptions, observability of outbound callout failures, concurrency and database CPU. The corresponding features all require a paid plan: `EM_Ept` ("EPT analysis"), `EM_SlowSoql` ("Slow-SOQL analysis"), `EM_UnhandledElf` ("Unhandled-exception ELF") and `Callout_Silent_Failures` ("Silent callout failure detection (error inside an HTTP 200)").

Reading the Event Log Files goes through the `OS_SelfOrg` self-callout, as does `Callout_Silent_Failures`: the procedure to wire it, then enable this tier, is in the Configuration Guide §3, “Wiring the `OS_SelfOrg` self-callout”.

With or without Shield

OrgGuardian does not require Shield, and draws no additional source from it. None of the sources one would expect (`TenantSecret`, `EncryptedData`, `ShieldPlatformEncryptionPolicy`, `EncryptionKeyMetadata`) is readable from Apex: `TenantSecret` is not API-accessible, the other three do not exist for Apex, and `FieldDefinition` carries no `IsEncrypted` field. The product therefore claims to audit **neither** the key lifecycle, **nor** the encryption policy, **nor** the deterministic/probabilistic scheme.

The only source actually available is the field describe, which states whether a field is encrypted: it answers on **any** org, with or without Shield, and applies equally to Shield and to classic encrypted text. The encryption posture reported by the product rests on it (**Encryption at rest for regulated data** feature, `Security_Encryption`, paid plan, on by default).

---

9. Retention and purge

The `OrgGuardian Data Retention` job (nightly) applies the retention periods from the general settings register:

`Retention_Finding_Days` · `Retention_Resolved_Days` · `Retention_Metric_Days` · `Retention_Log_Days` · `Retention_Snapshot_Days` · `Retention_Compliance_Days` · `Retention_Prediction_Days` · `Retention_Posture_Report_Days`

On the `FREE` plan, the purge caps every one of these windows at 7 days, whatever value the register holds, except `Retention_Posture_Report_Days`: the board report archives keep their configured window on every plan (the `FREE` plan writes no new one; those of the trial remain the subscriber's records). The product does not restore what the purge removes, including after a move to a paid plan.

Small org. On an org whose data storage quota is 20 MB or less (a Developer Edition, for example), `Retention_Log_Days` defaults to 7 days instead of 30: technical logs (`System_Log__c`), scan history (`Scan_Run__c`) and callout logs (`Callout_Log__c`). The purge reads the maximum quota (`DataStorageMB`) every night; a production org, whose quota counts in GB, keeps exactly its window. A saved `Retention_Log_Days` value always wins: on a small org, the **General settings** pane shows 7 as the default, and entering 30 there saves it. When the user who runs the purge cannot read the quota, the purge keeps 30 days and says so in the log (category `RETENTION_STORAGE_UNREADABLE`, one line a night); on the free plan, whose cap decides whatever the quota, it writes nothing.

The repeat filter and the alert cooldown never read back more than 7 days of log: the purge removes no line a reminder needs. Other readers look further back. All of them are `PRO` features: on a small org, they run only during the full-plan trial or under a subscription, and once enabled (only **Alert analytics** is enabled by default):

- escalation (`Alert_Escalation_Chains`): an unacknowledged step goes out again once its line is purged;
- flapping suppression (`Alert_Flap_Suppression`): an `Alert_Flap_Window_Minutes` window set above 3.5 days reads back two windows, more than the log kept, and can let an alert already reported go out again;

- owner reminders (`Remediation_Reminders`): a purged reminder counts as never sent in the queue order, and its owner moves back to the front;
- incident correlation (`Incident_Management`): for an incident whose correlation window began more than 7 days ago, the configuration log is stated partial (its window runs past the retention), never "no change", and the post-mortem freezes that state;
- the **Alert analytics** screen (`Alert_Analytics`): when the selected period is longer than the history kept, the header says so (*alert history kept: 7 days*) and the operational gain is computed over the days covered. The screen does not compare a period with a purged previous period. That rule holds on every org: with the default retention, only the shortest of the three periods shows a trend.

For escalation and flapping, that is one alert too many, never a silence. The org status screen states the scan history kept under the availability window (*Scan history kept: 7 days*) when it is shorter than the window; the value ledger and the incident drill record state the same window. On the free plan, these two screens state the 7 days its cap applies.

A quota read by the user, not by the org. Reading `DataStorageMB` depends on the user who reads it (measured: a system administrator reads it, a user holding only the `OrgGuardian_Admin` permission set does not). The purge reads the quota as the user who scheduled it and records each night the window it applied to logs and scan history. The org status screen, the value ledger and the **Subscription** screen state that recorded window whoever opens them, and **Subscription** gives the date of the run that recorded it. A record older than 3 days is no longer stated there: it signals a purge that stopped. Without a recent record, these three screens state a window only when it does not depend on the user who scheduled the purge: a saved `Retention_Log_Days`, the free plan, or a quota read above 20 MB by the user who opens the screen (the purge then keeps 30 days, whether it reads the quota or not). Otherwise they state *not measured*, whoever opens them, never 30 days nor 7. The purge records its window only when the user who scheduled it may write the package's protected settings ("Customize Application"): a purge scheduled with least privilege (`OrgGuardian_Manage_Features` alone, installation guide §5.3) never records anything, and these screens then state *not measured* on a small org. A purge that cannot read the quota keeps the window of a larger org, records it when it can, and the `RETENTION_STORAGE_UNREADABLE` log line flags it. The **General settings** pane, the incident drill record, the **Alert analytics** screen and incident correlation also use the window recorded by the purge. Without a recent record, when the window depends on the user who scheduled the purge, the first three state *not measured* (the **General settings** pane says so under the setting), and incident correlation keeps the shorter of the two windows, 7 days, and states its source as partial. For a window that depends on no one, save `Retention_Log_Days`.

A partial purge, by design. The purge runs in user mode, as the user who bootstrapped the scheduled jobs, and the package grants no permission set read or modify access to **all** the rows (`viewAllRecords`, `modifyAllRecords`) of an object. On the trace tables shared Public Read/Write (`Callout_Log__c`, `Scan_Run__c`, `Finding_Activity__c`, `System_Log__c`), that sharing grants editing a row, not deleting it: the purge deletes the rows that user owns, all of them if that user holds "Modify All Data", and, depending on the role hierarchy, those of their subordinates (a platform rule, not measured). The rows of another owner outside the hierarchy (an integration user whose calls are instrumented, for instance) stay past the retention; the refusal is counted in the `RETENTION_PURGE` log. It is also named by a Low finding, one per object (source `OrgGuardian:RetentionRefused`, target: the object's API name): the number of rows refused over the night and the cause (rows of other owners, or the code of another

error). Before deleting anything, the purge checks that its user holds Delete on the object; without it, nothing is tried and the same finding says so. The finding closes by itself, without a verification date, at the first purge of that object with no refusal; its playbook says how to let the purge through. The board report archives (shared read-only) follow the same rule. Setup → **Scheduled Jobs** ("Submitted By" column) shows the user who scheduled each job; if that user holds "Modify All Data", as a system administrator does, the purge is no longer partial. Two blind spots of the finding: a night when a batch of an object's purge stopped on an error the code cannot catch (a platform limit) does not close it, and the expired analyses (`Insight__c`), deleted all or nothing at the end of the chain, have neither the check nor the finding (an error there is only logged, under `RETENTION_INSIGHT`).

Two points to know:

- The purge **deletes** `Finding_Activity__c` in its dedicated phase.
- It **never touches** `Attestation__c`: compliance attestations are in no purge phase.

The purge's progress is stored in `OrgGuardian_PurgeCursor__c`. If this progress cannot be written, the Big Object phase concerned purges nothing that night: an ERROR line `RETENTION_CURSOR_NOT_SAVED` in the system log (`System_Log__c`) names the phase, and the next night replays the same window. No screen shows this state: it is read in the log.

---

10. Where to go next

- Who is allowed to do what: Permissions and Personas guide.
- Setting thresholds, channels, frameworks: the Configuration Guide.
- An empty screen, a missing job, a silent channel: the Troubleshooting Guide.
- What the product never does: the Security FAQ.
- Vocabulary: the Glossary.
- What the product reads and from where: the publisher's data source matrix, available on request.
- Enabling an optional capability (Event Monitoring, Tooling API, Heroku option, channels): the publisher's activation notice, available on request.
- Overall architecture: the product's architecture note, available on request.