Configuration Guide

OrgGuardian customer guide — no-code configuration: where it is done and what it writes.

> **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 (`License__mdt` is `orgguardian__License__mdt` there).

---

1. The principle: configuration lives in Custom Metadata Types

OrgGuardian is not configured through code, nor through data records: it is configured through **Custom Metadata Types** (CMT) that it owns, all in the `orgguardian` namespace.

The package ships **37** of them, which fall into two families.

The 26 types an administrator edits from a screen

When an administrator saves from a no-code editor, the package **deploys Custom Metadata records** through the Metadata API — never by DML, which the platform does not allow for Custom Metadata records. These editors, and the rule pack import described in §2, are the only places where the package writes Custom Metadata. A save is asynchronous: the change takes effect once the deployment has completed, a few moments later.

The table of the Settings screen in §2 names the type or types that each editor writes. The publisher's Metadata API notice for subscribers, which details what writes each of these 26 types, is available on request. The one type that table does not name, `Fleet_Policy__mdt` (the declared fleet policies), is written by the fleet policy editor, opened from the **Fleet** screen (**Fleet policies**, offered to administrators only).

The 11 types shipped read-only

The 11 types that have **no editor**: the package ships them, reads them, and never writes to them.

Type	What it holds	Fields
`License__mdt`	the org's licence plan	`Tier__c`, `Org_Id__c`, `Expires_At__c`
`Posture_Sla__mdt`	posture SLA targets	`Target_Health_Score__c`, `Target_Scan_Availability_Pct__c`, `Evaluation_Period_Days__c`

Type	What it holds	Fields
`Benchmark_Cohort__mdt`	health score quartiles per cohort *(to be computed on health scores as described in "How the health score is computed")*	`Cohort__c`, `P25_Score__c`, `P50_Score__c`, `P75_Score__c`
`Escalation_Policy__mdt`	escalation policy by severity	`Min_Severity__c`, `Route__c`, `Is_Active__c`
`Gate_Label__mdt`	feature switch labels, per language (empty = English, the master text)	`Gate_Name__c`, `Display_Label__c`, `Language__c`, `Active__c`
`Org_Insight__mdt`	insight texts	`Insight_Key__c`, `Title__c`, `Body__c`, `Language__c`, `Active__c`
`Attack_Path_Rule__mdt`	declared attack paths: steps which, observed together on the same principal, link an entry point to an asset	`Path_Label__c`, `Entry_Label__c`, `Asset_Label__c`, `Principal_Kind__c`, `Link_Keys__c`, `Order__c`, `Require_Id_Token__c`, `Severity__c`, `Active__c`
`Data_Jurisdiction__mdt`	mapping between a domain or an instance and a jurisdiction (country, GDPR adequacy), established without an outbound call	`Match_Kind__c`, `Match_Value__c`, `Country_Iso__c`, `Adequacy__c`, `Confidence__c`, `Evidence_Source__c`, `Is_Active__c`
`Parity_Rule__mdt`	settings that must be identical between the UAT sandbox and production	`Config_Type__c`, `Config_Key__c`, `Detail_Level__c`, `Ignore_Settings__c`, `Rationale__c`, `Order__c`, `Severity__c`, `Active__c`
`Release_Calendar__mdt`	calendar of Salesforce's seasonal releases, maintained without an outbound call	`Api_Version__c`, `Release_Name__c`, `Ga_Date__c`, `Local_Ga_Date__c`, `Ga_Window__c`, `Min_Supported_Api__c`, `Is_Active__c`
`Remediation_Effort__mdt`	estimated effort to close a source's findings, which orders the remediation plan	`Source_Prefix__c`, `Effort_Hours__c`, `Effort_Basis__c`, `Effort_Confidence__c`, `Requires_Change_Window__c`, `Active__c`

> `License__mdt` is a **protected** type, visible only to the package's code: a subscriber cannot grant itself a higher plan by deploying a record into it.

---

2. Where configuration is done in the interface

The app and its tabs

The package ships **one** Lightning app, **OrgGuardian**, and **43 tabs**.

Eight tabs are administration screens: the `OrgGuardian_Viewer`, `OrgGuardian_Analyst` and `OrgGuardian_Auditor` permission sets hide them, and only `OrgGuardian_Admin` shows them:

Tab	Label	What it is for
`OrgGuardian_Config`	Settings	all the no-code editors (see §3), and the rule packs (Custom rules section)
`OrgGuardian_ChannelSettings`	Alert channels	channel consent and testing
`OrgGuardian_AlertRouting`	Alert routing	routing rules
`OrgGuardian_Connections`	Connections	outbound connections declared in the org
`OrgGuardian_Enablement`	Enablement	getting-started wizard
`OrgGuardian_Plans`	Subscription	features unlocked by the licence plan, and the **Muting** section (§3)
`OrgGuardian_Cat_Admin`	Administration	the administration *category* tab, with two pills: **Getting started** opens Subscription, Enablement and Incident drill; **Settings** opens the editors of the Settings tab
`OrgGuardian_IncidentDrill`	Incident drill	incident drill (DORA art. 11): replays the real escalation chain on a scenario and returns a timestamped report; each run writes an audit record in the org, and live mode really notifies the on-call rota

The **35 other** tabs are viewing or working screens, visible to all permission sets. Three of them are **category** tabs that group the others: **All monitoring screens**, **All security screens** and **All executive screens** — the fourth, **Administration**, is the one in the table above, reserved for `OrgGuardian_Admin`.

The Settings screen and its 23 panes

The **Settings** tab opens the **OrgGuardian settings** page, which exposes **23 panes**. Twenty are editors that write one or more Custom Metadata Types; **Tooling API access** writes no setting of its own: it switches on, one at a time, the features that the self-callout serves, through `Feature_Gate__mdt`, like **Feature activation** (“Wiring the `OS_SelfOrg` self-callout”, §3); **Heroku option** and **Secrets and consents** write protected custom settings.

Pane (screen label)	Type(s) written
Feature activation	`Feature_Gate__mdt`
General settings	`Org_Setting__mdt`
Per-metric thresholds	`Monitor_Threshold__mdt`
Cost of risk	`Risk_Weight__mdt`
Compliance frameworks	`Compliance_Framework_Setting__mdt`
Branding	`Branding__mdt`

Pane (screen label)	Type(s) written
Detector tuning	`Detector_Setting__mdt`
Custom rules	`Detection_Rule__mdt`
Business criticality scopes	`Criticality_Scope__mdt`
Endpoint tagging rules	`Endpoint_Tag_Rule__mdt`
Maintenance windows	`Maintenance_Window__mdt`
Detection window	`Scan_Window_Policy__mdt`
Escalation chains	`Escalation_Chain__mdt`
Channels and templates	`Alert_Channel_Config__mdt`, `Email_Alert_Template__mdt`
Routing rules	`Alert_Routing_Rule__mdt`
Quiet hours	`Recipient_Delivery_Pref__mdt`
Posture digest	`Digest_Config__mdt` and `Report_Template__mdt` (the templates of the reports by recipient)
Risk accepted	`Acceptance_Policy__mdt`
Sensitive identities	`Sensitive_Principal__mdt`
Remediation playbooks	`Remediation_Playbook__mdt`, `Posture_Narrative__mdt`, `Posture_Recommendation__mdt`, `Root_Cause_Rule__mdt`
Heroku option	*(not a CMT: the protected custom setting `OrgGuardian_Heroku__c`)*
Tooling API access	*(no setting of its own: switches features on through `Feature_Gate__mdt`)*
Secrets and consents	*(not a CMT: the protected custom setting `OrgGuardian_Channel_Settings__c`)*

A second write path exists for three of these types: the **Export / Import rules** card of the **Settings** tab (**Custom rules** section) imports a rule pack (**Import rules**), which deploys `Acceptance_Policy__mdt`, `Detection_Rule__mdt` and `Detector_Setting__mdt` records. The details are in the Metadata API notice, available on request.

A cleared optional field is removed. In the following panes, clearing an optional field of an existing record removes it, and the record then follows the default the product applies to an empty field: **Routing rules**, **Escalation chains** (channel type, severity floor, delay, recipient permission set), **Posture digest** (blocks and recipient permission set of a report template), **General settings** (the `Api_Exfil_Allowlist` list; **Reset to default** removes the saved value), **Per-metric thresholds**, **Endpoint tagging rules**, **Branding**, **Detector tuning** (threshold, severity override), **Custom rules** (message template), **Business criticality scopes** (value of an inactive declaration), **Channels and templates** (subject, body, recipients, sender and reply-to address of an email template), **Maintenance windows** (scope, days), **Quiet hours** (time zone, bypass severity), **Detection window** and **Risk accepted**. Clearing a required field makes the save fail with a message. The other panes that write Custom Metadata have no optional field that an empty value would leave in place: their fields are required, or rewritten on every save.

Who is allowed to save

Two checks apply, one after the other, and a save needs both.

1. **OrgGuardian's own check**, before anything is written. It accepts a user who holds the product permission `OrgGuardian_Manage_Features` (“OrgGuardian: manage features”, carried only by the `OrgGuardian_Admin` permission set) or the platform privilege “Customize Application” or “Modify Metadata Through Metadata API Functions”. A check that cannot be performed counts as a refusal: it never grants the save. When this check refuses a Custom Metadata save, the user reads: *Denied: saving this configuration requires the Salesforce “Modify Metadata Through Metadata API Functions” permission, which System Administrators have. Neither “Customize Application” nor the “OrgGuardian: manage features” permission, even together, is enough. Nothing was saved.* This message names the right that the second check requires, not the three that this first check accepts. A holder of `OrgGuardian_Admin` always passes this first check: the refusal they can meet comes from the second one (item 2), which is also shown as a refusal that names that permission.
2. **The platform's own check**, on the write itself, which OrgGuardian does not bypass. The twenty Custom Metadata panes, the feature switches of the **Tooling API access** pane and the rule pack import save through a Metadata API deployment, and the platform requires the user who saves to hold “Modify Metadata Through Metadata API Functions”, which System Administrators have. “Customize Application” on its own is not enough: Salesforce refuses the deployment before it is queued, and nothing is saved. The pane then shows its refusal message, which names that permission, and the refusal is logged as an authorization refusal; a failure message that names no right (for example “Writing the configuration failed.”) signals instead a technical incident for a user who holds that permission, with the details in the debug log. The **Heroku option** and **Secrets and consents** panes also write protected custom settings, as does the mute of §3: that write needs “Customize Application”, which `OrgGuardian_Admin` does not grant. On a paid plan (or during the plan's trial), the activation assistant of the **Heroku option** pane also switches on `Heroku_Tier` through the same Custom Metadata deployment as the panes, which therefore needs “Modify Metadata Through Metadata API Functions” as well.

The product permission does not replace these platform privileges: on its own, it gets a user past the first check only. `OrgGuardian_Admin` grants none of them, so a user who holds that permission set, without such a privilege from a profile or another permission set, cannot save the Custom Metadata records or protected settings that these panes write, nor set a mute.

Personal alert preferences (the `Alert_Self_Service` feature, off by default). The **My alerts** screen saves them in a protected custom setting, in user mode like every other write: saving needs “Customize Application”, which `OrgGuardian_Admin` does not grant. A user without it reads their own preferences but cannot save them; an administrator who holds that privilege sets them for each recipient from the **Team preferences** section of the same screen. OrgGuardian does not bypass user mode for this write.

The trace of a save. Each Custom Metadata record that an OrgGuardian pane or a rule pack import deploys leaves one line in the package's internal log, `System_Log__c`, category `CONFIG_CHANGE`: `Configuration changed (job <id>): <Type>.<Record>` once Salesforce has applied the deployment, or `Configuration change NOT applied (job <id>): <Type>.<Record>` if it failed; `<id>` is the deployment ID, the one shown on the Deployment Status page in Setup. The line is written in the name of the user who saved, when the deployment completes; it names the record written, never the value. One line per

record actually written. The general settings and the risk-cost weights write only the values changed, the bulk feature toggle skips features already in the requested state, and a weights or alert-channel request that changes nothing (an unchanged grid, a channel set back to the state it already has) writes nothing, deploys nothing and says so. The other panes rewrite the record they save, even unchanged, and a rule pack import redeploys each of its valid entries: one line each. The very first activation of an alert channel also writes, one line each, the default channels that have no record yet: without them, they would stop alerting. The protected settings that the **Heroku option** and **Secrets and consents** panes write, like mutes, leave no `CONFIG_CHANGE` line; the `Heroku_Tier` switch that the activation assistant turns on leaves one (`Feature_Gate__mdt.Heroku_Tier`). These lines follow internal logging (the `System_Logging` feature, on by default) and its retention (`Retention_Log_Days`).

---

3. The main settings

Detection thresholds

Two levels, from the most general to the most specific.

The general settings register — `Org_Setting__mdt`, edited in the **General settings** pane: one record per key, holding a single value: a number (`Number_Value__c`) for every key but one, `Api_Exfil_Allowlist`, whose text value (`Text_Value__c`) lists the legitimate connected apps that the API exfiltration detector leaves aside. Saving the **General settings** pane writes only the settings you changed: a setting that has never been saved keeps following the product's built-in default. Setting a value back to its default saves that value.

Some families among the keys:

Family	Keys (value of `Org_Setting__mdt.DeveloperName`)
Limit consumption levels	`Limit_Pre_Alert_Pct`, `Limit_Warning_Pct`, `Limit_Critical_Pct`
Posture score bands	`Score_Band_Green`, `Score_Band_Amber`
Intensity of one open finding on the risk heat maps, by severity	`Score_Penalty_Critical`, `Score_Penalty_High`, `Score_Penalty_Medium`, `Score_Penalty_Low`, `Score_Penalty_Info`
Alert noise	`Alert_Mass_Threshold`, `Alert_Cooldown_Minutes`, `Alert_Reminder_Hours`, `Alert_Flap_Threshold`, `Alert_Flap_Window_Minutes`, `Alert_Correlation_Min_Pct`
Data storage	`Storage_Guard_Pct`
Routing by severity	`Alert_Route_Critical`, `Alert_Route_High`, `Alert_Route_Medium`, `Alert_Route_Low`, `Alert_Route_Info`
Retention (in days)	`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`
Certificates	`Cert_Warning_Days`, `Cert_High_Days`
Business hours	`Business_Hours_Start`, `Business_Hours_End`

Family	Keys (value of `Org_Setting__mdt.DeveloperName`)
Code quality	`Quality_Coverage_Floor_Pct`, `Quality_Coverage_Medium_Pct`, `Quality_Coverage_High_Pct`, `Quality_Min_Api_Version`, `Quality_Stale_Api_Medium_Count`, `Linter_Weight_High`, `Linter_Weight_Medium`, `Linter_Weight_Low`
Privileged accounts	`Max_Modify_All_Data_Assignees`, `Max_View_All_Data_Assignees`, `Max_Author_Apex_Assignees`, `Max_Manage_Users_Assignees`, `Max_Modify_Metadata_Assignees`, `Max_View_All_Users_Assignees`, `Max_Manage_Encryption_Keys_Assignees`, `Max_View_Event_Log_Files_Assignees`, `Max_Customize_Application_Assignees`, `Max_Manage_Profiles_Assignees`
Financial visibility	`Financials_Visibility_Mode`, `Annual_SF_Spend_Eur`

The five settings `Score_Penalty_Critical` to `Score_Penalty_Info` tune **no score**: not the pack scores, not the org health score, not the points the remediation plan promises for each fix. They only tune the intensity of one open finding on the risk heat maps (the Risk heat map screen and the heat map of the Insights screen), and the panel says so under each of them. A score is the share of evaluated controls that passed, weighted by severity (see "How pack scores are computed" and "How the health score is computed" below): changing these five settings does not move it.

`Alert_Reminder_Hours` (24 hours by default, 1 to 168) sets the reminder of a Critical finding that stays open and unchanged; a High or Medium finding is reminded once a week. `Storage_Guard_Pct` (90% by default, 50 to 100) sets the threshold of the storage guard, above which the package stops its non-essential log writes (operations guide, alerts). 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, and the pane shows that default; a saved value wins (operations guide, section 9). The default the pane shows is the one the nightly purge applies, whoever reads the pane. When nothing the purge has recorded recently tells which of the two defaults applies, the pane says that the default is not measured, and a note under the setting gives both cases.

The editor rejects an inconsistent value before writing. Each key has an allowed range, and crossed pairs are refused: the green score band (`Score_Band_Green`) must sit strictly above the amber one (`Score_Band_Amber`), the certificate High threshold (`Cert_High_Days`) strictly under the warning period (`Cert_Warning_Days`), and the start of business hours (`Business_Hours_Start`) strictly before the end (`Business_Hours_End`). The three limit consumption levels are not compared with one another in this pane: keep them in increasing order.

Per-metric thresholds — `Monitor_Threshold__mdt`, one record per metric:

Field	Role
`Metric_Key__c`	the metric concerned
`Pre_Alert_Pct__c`	pre-alert level
`Warning_Pct__c`	warning level
`Critical_Pct__c`	critical level
`Active__c`	whether the rule is active

The **Per-metric thresholds** pane refuses a metric whose pre-alert, warning and critical levels are not in strictly increasing order. For its metric, an active record takes priority over the three general limit levels; whenever such a record is active, the **General settings** pane says so next to those levels. Clearing the pre-alert of a threshold that had one removes it: the metric then follows the general pre-alert level (`Limit_Pre_Alert_Pct`, 60% when it is not set), as the field says.

Per-detector tuning — `Detector_Setting__mdt`: `Detector_Key__c`, `Threshold__c`, `Severity_Override__c` (force a detector's severity), `Active__c` (switch it off).

Under **Declared false positives**, the **Detector tuning** pane also shows, read-only, the share of insights that your users dismissed as false positives because the threshold was too strict (for anomalies, also because the behaviour was expected), per insight type, over the window of the calibration advice (**General settings**: `Advisor_Window_Days`, "Calibration advice: depth of anomaly history and insight judgements analysed") capped by the insight retention, with the current value of the setting that a raise would target. From 4 judged trend insights and a share at least equal to the rate of the calibration advice (`Advisor_Noise_High_Pct`, "Calibration advice: noise rate beyond which the product advises desensitising"), the calibration advice recommends raising "Trends: change beyond which a metric is judged to be degrading" (`Trend_Degradation_Pct`). The advice is saved to the insight feed only when **Saved adaptive threshold recommendations** (`Intelligence_AutoTune`) is on. OrgGuardian never changes the setting itself.

Custom rules — `Detection_Rule__mdt` lets you create a detection specific to the org: `Metric_Key__c`, `Operator__c`, `Threshold__c`, `Severity__c`, `Category__c`, `Message_Template__c`, `Active__c`.

How pack scores are computed

The Security, Code quality, Callouts and Monitoring packs each publish a score from 0 to 100. It says **what share of what the last scan checked is compliant**, weighted by severity.

- **A control** is a detector of the public detector catalogue, attached to the pack of its category. All the findings of one detector form a single control: eight standard objects shared publicly are one failed control, not eight.
- **Its weight** follows its severity: Critical 10, High 5, Medium 2, Low 1. A purely informational control weighs 0 and does not count; an **Info** notice makes no control fail.
- **A control fails** when it has at least one open finding that no decision of your team covers: it then weighs the severity of its worst such finding. **It passes** when it has none: it then weighs its nominal severity, the highest one the catalogue gives it.
- **A control with no finding counts as passed only when its read was proven complete during the scan**: its source was read in full, with no truncation, no refusal from the row budget and no cut in the writes. The absence of a finding is not enough.
- **A control that is not evaluated does not count**, neither as passed nor as failed: its feature is off, your plan does not include it, it runs outside the scan, its collector cannot prove a complete read yet, or its read was partial during this scan. A control that has an open finding is always counted: a partial read removes passes, never failures. One exception, your plan: a control it does not include is not evaluated, open finding included (see "How the health score is computed").
- **The score** is 100 × (sum of the weights of the passed controls) / (sum of the weights of the evaluated controls), rounded to the nearest integer. It is 100 only when no control fails and no decision is in force, and 0 only when no control passes.

- **Your team's decisions** (acknowledgement or snooze in force, risk accepted one by one, finding marked Ignored, active acceptance policy `Acceptance_Policy__mdt`) count the control as passed, but apart: the screen says how many controls passed by decision, and the score is then capped at 99. A decision is credited only on a control that is still read: its feature on, inside the scan and in your plan; otherwise the control is not evaluated. Active acceptance policies count in the pack scores as in the health score.
- **Nothing evaluated**: the pack shows "Not measured", never 0 or 100.
- **Low coverage**: when fewer than a quarter of the pack's controls are evaluated, or when the pack's read is partial, the pack carries the "Partial" mark instead of green.
- **What moves the score**: fixing an evaluated control raises it; a new failure or a worse one lowers it; fixing a single instance moves it only when the worst open severity of the control goes down. The score can also move without a fix when the set of evaluated controls changes (a read complete again, a feature switched on): the displayed change is then marked "Coverage changed".
- **An open Critical finding** of the pack, whatever it is (attack paths and correlated incidents included, snoozed or accepted included), keeps the pack red, even with a high score.

The colour bands (`Score_Band_Green`, `Score_Band_Amber`) apply to pack scores as well. The scores of a single collector (org hardening, Apex and Visualforce lint, technical debt) keep their own calculation.

How the health score is computed

The org health score, the large figure of the Home page, of the Dashboard screen and of the Executive summary, follows the model of the pack scores: **the share, weighted by severity, of the controls evaluated by the last scan that passed**, this time over **every control of the catalogue**: those of the four packs, plus those of logins (Logins category) and of the org's limits (Limits category), which no pack scores. It moves with fixes, like the pack pills.

- **Every control, same weights.** Each detector of non-zero weight in the public detector catalogue is a control, Critical 10, High 5, Medium 2, Low 1. Limit usage is not a separate term: it is the org limits control, which fails at the weight of its worst finding (Medium at 60%, High at 75%, Critical at 90% consumption, by default).
- **Same proof.** A control with no finding counts as passed only when its read was proven complete during the scan; a control that is not evaluated does not count; a control that has an open finding is always counted. A partial read removes passes, never failures.
- **Your plan.** A control your plan does not include is **not evaluated, open finding included**. When the 30-day trial ends, no scan reads the controls of the paid features any more: their findings still open no longer lower the score, and its evaluated controls shrink to those of the free plan. A feature your team switches off, by contrast, does not remove its failures: an open finding of a detector that is off remains a failure. The same rule applies to the pack scores. The Security pack also runs on the free plan, for five families only: human accounts without MFA, guest user exposure, public file links, connected apps and OAuth tokens, holders of "Modify All Data" and "View All Data". Its score there is computed on the security controls of the free plan, including those of this core and the Health Check settings OrgGuardian reads, and its low coverage marks it "Partial", never "Healthy"; the public detector catalogue says which detectors run on the free plan.

- **Your team's decisions** count the control as passed, apart, and cap the score at 99: acknowledgement or snooze in force, risk accepted one by one, finding marked Ignored, active policy of the register (`Acceptance_Policy__mdt`). A decision is credited only on a control that is still read (feature on, inside the scan, in your plan).
- **When it is published.** The health score is recomputed only when **every pack your org is entitled to** has published its score during the scan. An auxiliary step that fails (connectivity probe, Health Check…) does not block it: its controls are simply not evaluated. A pack that fails leaves it at its previous value and date, and the **Since the previous scan** narrative of the **Dashboard** screen dates the score ("as computed on …") and flags a partial read.
- **"Not measured"**: no control evaluated, or no scan has yet been able to publish the health score (a new install where a pack fails); never 0 or 100.
- **"Partial"**: when the score would be in the green band but the read is partial (fewer than a quarter of the catalogue's controls evaluated, a read notice on the Security screen, a partial read or low coverage of the Security, Code quality or Monitoring pack), the word under the ring is "Partial", not "Healthy". Low coverage of the Callouts pack alone does not trigger it: it evaluates few controls by construction at install.
- **An open Critical finding** (snoozed or accepted included, Ignored excluded) keeps the word "At risk", whatever the score.
- **What moves it**: as for a pack score. A change whose evaluated set has changed is marked "Coverage changed", and the score drop alert (`Score_Drop_Alert_Threshold`) is not sent when passes have left the evaluated set since the previous scan (partial read, feature switched off, end of trial); otherwise, a drop caused by new failures is alerted.

What reads this score. Board report archives, once sealed, keep the value of their date. A custom rule that targets `Org:HealthScore` reads this score (operations guide). The **Cohort posture benchmark** feature (off by default, paid plan) compares it with the P25, P50 and P75 quartiles of a cohort you publish yourself as a `Benchmark_Cohort__mdt` record (Setup → Custom Metadata Types): the package ships none, and until you publish one nothing is shown. Compute those quartiles on health scores of this model, never on another scale, before reading the comparison of the Executive summary. The Heroku tier, when enabled, judges each summary with the baseline in force at its receipt date, and prints the version of that baseline and its date on the page.

Compliance frameworks

The package ships a **compliance mapping** as a static resource, `OrgGuardianComplianceMap` — version `2026.10.2`, **126 entries**, covering **8 frameworks**: `DORA` (19 entries, 7 distinct articles), `NISTCSF` (17, 8), `NIST80053` (17, 13), `NIS2` (15, 7), `SOC2` (15, 10), GDPR (framework code `RGPD`; 15, 9), `ISO27001` (14, 10) and `SOX` (14, 12) — 76 distinct framework × article pairs in total. Deviations from the approved baseline (`Security:BaselineDrift`) are mapped, framework by framework, to the article of configuration drift. No entry declares its pack score as positive evidence (`Security:Score`, `Callouts:PostureScore`, `Monitoring:PostureScore`, `CodeQuality:PostureScore`): a score reads 100 when the pack finds nothing, so it proves no facet. A control reads "Covered" only when **each** of its facets produced its own evidence (a mapped finding, or the metric of its detector); otherwise it stays "Not Evaluated", and the row names the facets with no evidence (the CSV export carries them in its last two columns, `ProvenBy` and `UnprovenFacets`). The 30 catch-all entries of a pack that count in coverage (bare prefix, `Monitoring:` for instance) are proven only by a mapped finding; the screen names them "Other Monitoring pack checks", never by the name of the whole pack. Facets with no evidence are named

only when their absence was measured: under a refused read, a capped census or stale metrics, the freshness reason says enough. A custom map that declares a pack score is read the same way: the score is ignored, behind a prefix key too (`Security:` is not proven by `Security:Score`).

The administrator chooses, org by org, which ones are in scope, in the **Compliance frameworks** pane: `Compliance_Framework_Setting__mdt` with `Framework__c` (the framework code) and `Enabled__c`. A deselected framework disappears from the compliance screens.

The mapping itself can be **refreshed without a package upgrade**. From the same pane, an administrator imports a mapping file validated by their compliance team; OrgGuardian stores it as a Salesforce Files document titled exactly `OrgGuardianComplianceMap-override` and applies it in place of the shipped mapping. Validation is all-or-nothing: an invalid file is never applied, not even partially. Files are used rather than Custom Metadata because the mapping exceeds the size limits of a CMT record. The imported document is **shared with the organization as read-only**, so that every user, the scheduled scan and the board report apply the same table, and the Id of its version is **recorded in the protected setting** `OrgGuardian_Compliance_Map__c`. Importing and reverting to the shipped mapping require the **Customize Application** permission: without it, the platform refuses the record and the import is cancelled. Only the recorded version applies: a file with the same title, private or shared by someone else, is neither applied nor deleted. Reverting to the shipped mapping clears the record and touches only the recorded document, never another document bearing that title: it deletes it when you own it, and otherwise withdraws its share with the organization.

Risk weighting

`Risk_Weight__mdt` holds the valuation: `Severity__c`, `Category__c`, `Amount_Weight__c`. The **Cost of risk** pane writes two sets of records — the weight per severity × category pair, and the cost per severity. The package ships no tariff: every weight is zero until an administrator sets it. Saving writes only the amounts you changed: a weight never saved keeps following the product's default, and refining a category writes only the severities whose amount differs from the one in effect; a severity left at its inherited amount keeps following the per-severity cost. With no amount changed, the save button stays inactive, and a request that carries none is refused without deploying anything.

Two modulation settings live in the general register: `Risk_Occurrence_Step_Pct` and `Risk_Occurrence_Max_Extra`.

⚠ The euro calculation is **off by default**: the **Cost of risk** feature requires a paid plan and ships switched off (its state is shown on the **Feature activation** screen). Without a licence and without enablement, no amount is produced. See §5 of Permissions and Personas.

Criticality scopes

`Criticality_Scope__mdt`, edited in the **Business criticality scopes** pane of the Settings screen (the **Business criticality scopes** feature, `Criticality_Scoping`, paid plan, off by default) — one scope per row. A scope raises or lowers the severity of the findings it matches, from their next scan:

Field	Role
`Dimension__c`	what the scope looks at: `Target` (the object, class, connected app or identity a finding is about), `BusinessUnit` (the Division of the user a finding is about, recorded by four identity checks only: MFA gap, toxic combinations, effective permission drift, privileged account hygiene), `Source` (the check that raised it) or `Category` (its pack)
`Match_Value__c`	the value matched, case ignored; a star at the end, at the start, at both ends, or a single one in the middle; any other use of stars, and a lone star, is refused
`Tier__c`	`Critical` (two notches up), `Important` (one up), `Standard` (none, to exempt a narrower scope from a wider one), `Secondary` (one down); the result stays between Info and Critical
`Active__c`	whether the scope is active (the editor does not delete a scope: unticking it takes it out of the calculation)

The narrowest scope wins: a target beats a business unit, which beats a source, which beats a category, and an exact value beats a pattern. **A business unit scope can only raise a severity** (tiers `Critical` and `Important` only, and only beyond the scope that would win without it), because a user's Division has no field-level security: anyone who can edit that user's record can change it, which may include the user themself. A user who leaves a unit falls back to the policy that applies without it. A scope created by the editor gets an opaque API name (`Scope_` followed by sixteen hexadecimal digits): the name you type lives in its label only.

A finding whose severity a scope moved keeps the severity its check reported (`Finding__c.Base_Severity__c`) and the API name of the scope (`Finding__c.Criticality_Scope__c`); the finding sheet says so, with the scope's label for the readers of the finding's target only (never for the `OrgGuardian_Viewer` set, which is told that a scope lowered or raised the severity, without its name). Each kind of lowering of a High or Critical finding is written to the log once per scan transaction (`CRITICALITY_SCOPE_LOWERED`, WARN: the scope's dimension and tier, the severities before and after and the finding's category, without the target or the scope), not one line per lowered finding. **Condition**: this trace is written, and cleared, only when the user who runs the scan can write these two fields (`OrgGuardian_Admin`); otherwise the severity is moved without a trace, a trace written earlier can outlive the scope that justified it, and one WARN line `CRITICALITY_SCOPE_TRACE` per scan transaction (each pack runs in its own) says so while the feature is on. Likewise, the business unit is recorded only when the account running the scan can write `Finding__c.Business_Unit__c`: otherwise it is neither set nor cleared, and one WARN line `BUSINESS_UNIT_NOT_MEASURED` per scan transaction says so — a business unit that was not measured is never shown as empty. The remedy is to assign `OrgGuardian_Admin` to the scan account (it grants write access to the three fields). `User.Division` has no field-level security: every user who reads users reads it, and there is no read access to grant. The Division is read only when an active `BusinessUnit` scope or an active routing rule that names a business unit needs it; otherwise the identity checks do not read it.

Alert routing

`Alert_Routing_Rule__mdt`, edited in the **Routing rules** pane — one rule per row:

Field	Role
`Min_Severity__c`	minimum severity that triggers the rule
`Category__c` / `Finding_Type__c`	filter on the nature of the finding
`Channel_Type__c`	target output channel
`Route__c`	destination
`Recipient_Permission_Set__c`	recipient group, designated by a permission set
`Priority__c`	evaluation order
`Is_Active__c`	whether the rule is active
`Criticality_Tier__c`	criticality tier of the finding's winning scope; optional
`Business_Unit__c`	business unit (Division) of the user the finding is about; optional

The **Routing rules** pane reads, shows and writes the last two fields like the other fields: the business unit is compared whole (a star is not a wildcard there, and the pane refuses it), 80 characters at most; the tier is one of `Critical`, `Important`, `Standard`, `Secondary`; a rule that names a business unit and suppresses the alert is refused when saved, and the pane also refuses to **activate** one set in Setup (as it does for a starred business unit); deactivating it stays possible. Clearing an optional field of an existing rule in the pane (finding type, severity, route, priority, recipient permission set, business unit, tier) removes it from the rule. A **finding-type-only** rule (category left empty) is saved with the `*` category, which the type narrows; set in Setup with an empty category, a rule matches no finding, even with a type, until the pane saves it or toggles it again. Creating a rule whose name is that of an existing rule is refused: open the rule to change it. **My alerts** shows the business unit and the tier for each rule that reaches you. While the **Business criticality scopes** feature is off, a rule that names a tier matches no finding: the pane and **My alerts** show *"(not active)"* after the tier, **My alerts** explains it once above the list, and the form says so as soon as a tier is chosen. Each of the two fields adds a point of specificity to the rule. The tier is that of the finding's WINNING criticality scope (“Criticality scopes”, above), none while the **Business criticality scopes** feature is off; the business unit is compared ignoring case and surrounding spaces. **They apply only to the alert about ONE finding**, never to the digest nor to a grouped alert, which summarise several subjects. **A scope rule does not confine**: a finding it does not match falls back to the next matching rule, usually a wider one — the case of any finding from a check other than the four that record a business unit (login anomalies and failed logins, identity verification, dormant administrators of the posture check, silent principals, access of deactivated accounts, connected-app token holders, and any check that is not about a user), of an empty or unmeasured Division, and of an unknown tier. **A rule that would silence a dispatch in the name of a business unit is ignored** (one WARN line `ALERT_ROUTING_RULE_IGNORED` in the log), and a rule that silences a dispatch never matches a tier won by a business unit. **A business unit rule ADDS its audience, it never replaces it**: for the alert about one finding, the rule that wins without regard to the business unit (or, without a rule, the default channels) applies as if no business unit rule existed, including when it suppresses the alert; the rule that wins once the business unit is taken into account, if it differs — a rule that names a business unit, a tier rule that a business unit makes win, or a broader rule that takes the

finding over when that tier sets the general rule aside — sends IN ADDITION, through its channel and to its recipient permission set. A channel goes out once, and each recipient gets one message. Such a rule whose channel does not deliver (email inactive, channel feature off) adds nothing and takes nothing away. The subject of a finding, even if they can change their own Division, can neither silence nor divert the alert about them: they can only have it sent, in addition, to another audience you configured.

In addition: `Escalation_Chain__mdt` (`Step_Number__c`, `Delay_Minutes__c`, `Min_Severity__c`, `Channel_Type__c`, `Recipient_Permission_Set__c`, `Is_Active__c`) for stepped escalation, in the **Escalation chains** pane, and `Recipient_Delivery_Pref__mdt` (`Permission_Set__c`, `Quiet_Start_Hour__c`, `Quiet_End_Hour__c`, `Timezone__c`, `Bypass_Severity__c`, `Is_Active__c`) for quiet hours per group, in the **Quiet hours** pane: each group's quiet hours apply in that group's time zone, with a severity that bypasses them.

⚠ The **Alert routing** feature ships switched off and requires a paid plan: configuring a rule is not enough; the feature must also be enabled.

Digest configuration

`Digest_Config__mdt`, edited in the **Posture digest** pane — two fields only: `Cadence__c` and `Is_Active__c`. The digest is the **only** scheduled job in the product whose frequency depends on a subscriber setting. A new cadence is applied to the "OrgGuardian Posture Digest" job (**Setup → Scheduled Jobs**) the next time an entitled user opens the OrgGuardian **Home** tab; until then, the **Posture digest** pane states that the chosen frequency is not yet applied.

The email template and recipients live in `Email_Alert_Template__mdt`, edited in the **Channels and templates** pane: `Subject_Template__c`, `Body_Template__c`, `Recipient_Type__c`, `Custom_Recipients__c`, `Sender_Display_Name__c`, `Reply_To__c`, `Severity_Filter__c`, `Is_Active__c`.

The **Posture digest** pane also holds the **reports by recipient**, and the digest can carry the **remediation register**:

- `Report_Template__mdt` chooses, per recipient — executive management (`DIRECTION`), CISO (`RSSI`) or administrator (`ADMIN`) —, the blocks of the report and their order (`Sections__c`). Three more fields apply to the executive and CISO templates: the recipient permission set (`Recipient_Permission_Set__c`, its API name), the frequency (`Cadence__c`: `DAILY`; `WEEKLY`, on Mondays, the default; `MONTHLY`, on the first Monday of the month) and, for the executive template only, the board pack (`Include_Board_Pack__c`). They act only with the **Reports by recipient** feature (`Report_Routing`, paid plan, off by default) and an active email channel, and, for the board pack, the **Board report** feature as well: the report is then emailed to the **active internal** members of the named permission set, at the night hour of the digest; each block goes in only if that permission set **itself** grants read access to the block's sources, and the board pack only if it reads the sources of the executive screen and, while the pack carries euro amounts in the "Permission required" mode, grants `OS_View_Financials` (the Permissions and Personas guide §3). An `ADMIN` template is never routed: the administrator report remains the digest sent to the alert recipients. With the feature off, only the digest is sent, and the pane reads neither the audience nor the rights of any template; with the feature on, it reads them only for an active template that names a permission set, and shows the blocks that the send will withhold, computed on those the engine will actually compose.

- The body of a report, like the digest's, fits in **2,000 characters**: a block that does not fit is never cut; it is left out **whole** and named on the last line by a readable name, the block's title when the body prints one (“Not included, to keep this report within 2000 characters: Remediation register.”). A body that fits is sent as composed.
- The `REMEDIATION` block ends the default `ADMIN` and `RSSI` templates when the **Remediation reminders** feature (`Remediation_Reminders`, paid plan, off by default) and the **DORA / NIS2 pack** (`Pack_Dora`) are on: open gaps, overdue gaps (with the longest delay), gaps without an owner, then the three most overdue controls (framework, article, days overdue), never a name. It says “not measured” when the register cannot be read, and “partial count” when the register is too long to be read in full: the counts then cover only the entries read, and owners are not counted. This block travels **inside** the digest, through the digest's channels already enabled (email included) and to all of its recipients, literal addresses included, whose access to the register is not checked. With these features off, `REMEDIATION` is an unknown block name and the digest does not carry it. The same feature rings the **remediation reminders** (the Operations Guide §5, the Glossary §7).
- The cadence of `Digest_Config__mdt` does not decide the job's schedule alone: a **daily** report by recipient makes the “OrgGuardian Posture Digest” job run every night, and the weekly digest and the remediation reminders then keep their day, Monday. The pane says so next to the setting.
- By default, the digest, the reports by recipient and the SLA compliance summary are emailed in a **report** layout (a title, the body, the read-only reminder), not in the layout of a finding alert; an active `Email_Alert_Template__mdt` record still takes precedence.

Feature activation

`Feature_Gate__mdt`, written by the **Feature activation** pane — two fields: `Is_Enabled__c` and `Tier_Required__c`. It is the enablement lever, and it is **asymmetric by design**:

- **When no record exists** for a feature, the product applies that feature's built-in default: the state that the **Feature activation** screen shows after installation.
- **A record, when present, can switch a feature off** (`Is_Enabled__c = false` ⇒ off), or switch it on within what the plan allows.
- **A record cannot lower the required plan**: the plan applied is the **higher** of the product's built-in plan for that feature and the plan in the record. A subscriber can tighten, never self-grant.

There is also a **mute**, in the **Muting** section of the **Subscription** tab (stored in the protected custom setting `OrgGuardian_Feature_Mute__c`): it only **forces off** a feature that is already enabled; it never enables any.

Enabling in one go, and what never follows

The **Feature activation** screen offers three actions, from the broadest to the narrowest:

Action	Scope	What it enables
Enable everything possible	the whole org	everything your plan allows that involves no network egress and is not one of the two held inbound exposures, **Finding details in the REST API** and **Parity fingerprint API**
Enable this domain	one section (Alerts, Security & audit…)	the same, limited to that domain
a row's switch	one feature	that feature alone

The package declares **116 features**. **42 features are on by default** and **74 off**, the **22 outbound features** and the **2 held inbound exposures** among them. None of the 42 is an outbound feature or one of the two held inbound exposures; one of them, **REST API**, does serve the read-only status resource from installation (the Security FAQ §4). Another of the 42, **Health Check parity (SOQL, no outbound call)**, reads only through SOQL queries and, from the first scan, measures the four Health Check settings that can be recomputed this way. Several **in-org analyses** that make no outbound call are on by default — among them **Field exposure scanner (sensitive fields, FLS)**, **Data-export risk detection**, **Attack paths**, the configuration drift screen, the **User access review**, which reads the org and writes nothing, and the **Privilege matrix and consolidation plan** screen, which reads the org in its own transaction and writes nothing — each reserved to the plan that includes it. **Event Monitoring security events**, which reads nothing without your Event Monitoring license and makes no outbound call, is off by default: its queries have not yet read a real event in an org that holds Event Monitoring. An administrator turns it on from the **Feature activation** screen or the **Enablement** tab (**Enable everything possible** turns it on as well).

Bulk actions hold back the **22 outbound features** and the **2 inbound exposures** (**Finding details in the REST API** and **Parity fingerprint API**: REST routes that serve your data to an authenticated caller). They are not the only inbound surfaces: **Deployment advisor (CI check)** and **Incident management**, with its write, **REST write on incidents (PATCH)**, can still be enabled in bulk, and the **REST API** feature serves the status resource from installation. That leaves **50 features that can be enabled in bulk**: 2 on the `FREE` plan and 48 on a paid plan. This number is a **ceiling, not a promise**: “Enable everything possible” only ever enables what **your plan** allows. The button therefore shows **the actual count for your org**, and the report that follows the click states what was actually enabled. The **22 outbound features** stay out in every case, and require a decision feature by feature.

A bulk action never enables an outbound feature, nor one of the two held inbound exposures (**Finding details in the REST API**, **Parity fingerprint API**). The 22 outbound features fall into four families: five HTTP alert channels (Slack, Teams, webhook, PagerDuty, and the reading of Slack reactions, **Slack reaction acknowledgement**); four exports and integrations (the SIEM export, the ITSM integration, the automatic ITSM ticket creation, the Heroku option); seven external probes (endpoint reachability, certificate and TLS expiry, endpoint health check, multi-org aggregation, **Linked org findings** and **Fleet parity**, the two other reads of your linked orgs, and callout resilience); and six calls to your own org's API. They remain individual and consented to. Two reasons, and the second is as practical as the first: the “no outbound call by default” posture would be lost in a single click, and each of these features

requires, in any case, a URL, a Named Credential or a routing key that nobody can provide on your behalf — switching them on in bulk would produce features that fail silently. The **Chatter** channel is the exception and follows bulk actions: posting a feed item stays inside your org.

What the screen reports is measured, not assumed. Writing a feature switch goes through a metadata deployment, which is asynchronous and **atomic**: until it has completed, nothing is active, and a single rejected record cancels the whole batch. The screen therefore never reports the change as done when the call returns: it **re-reads the actual state** and reports what it found — "12 enabled" and, where applicable, "3 not applied yet". If that second line persists, use **Refresh** a moment later; if it still persists, the deployment was rejected and nothing in that batch was applied.

Scan and maintenance windows

- `Scan_Window_Policy__mdt`, edited in the **Detection window** pane: `Start_Hour__c`, `End_Hour__c`, `Business_Hours_Only__c`, `Dry_Run__c` — when scans are allowed to run, and whether a run is a dry run.
- `Maintenance_Window__mdt`, edited in the **Maintenance windows** pane: `Days_Of_Week__c`, `Start_Time__c`, `End_Time__c`, `Affected_Scopes__c`, `Suppress_Alerts__c` — the windows during which alerts are silenced.

Other shipped settings

Type	What it sets	Fields
`Acceptance_Policy__mdt`	accepted risk: approver, reason, expiry date	`Min_Severity__c`, `Category__c`, `Approver__c`, `Approver_User_Id__c`, `Reason__c`, `Expires_On__c`, `Dedupe_Key_Pattern__c`, `Recorded_By__c`, `Recorded_On__c`, `Active__c`
`Sensitive_Principal__mdt`	identities to watch closely	`Principal_Name__c`, `Active__c`
`Endpoint_Tag_Rule__mdt`	automatic classification of endpoints	`URL_Pattern__c`, `Provider__c`, `Service_Type__c`, `Suggested_Criticality__c`, `Is_Active__c`
`Branding__mdt`	white-labelling of reports	`Org_Display_Name__c`, `Logo_Url__c`, `Primary_Color__c`, `Footer_Text__c`, `Show_Powered_By__c`, `Is_Active__c`
`Remediation_Playbook__mdt`	steps to follow per finding family	`Step_01__c` … `Step_10__c`, `Business_Impact__c`, `Risk_Of_Change__c`, `Setup_Relative_Url__c`, `Source_Prefix__c`, `Language__c`, `Active__c`
`Root_Cause_Rule__mdt`	probable cause of an outbound call error	`Error_Type__c`, `Likely_Cause__c`, `Recommended_Action__c`, `Language__c`, `Active__c`
`Posture_Narrative__mdt`	text of the summaries	`Segment_Key__c`, `Prose__c`, `Language__c`, `Active__c`
`Posture_Recommendation__mdt`	recommendation associated with a rule	`Rule_Key__c`, `Action_Template__c`, `Severity__c`, `Language__c`, `Active__c`

In the **Endpoint tagging rules** and **Branding** panes, clearing an optional field of an existing record removes it: a tagging rule's provider, service type or suggested criticality; the display name, logo, color or footer of the `Default` branding record, which then fall back to the product defaults.

The same holds in the **Detection window** and **Risk accepted** panes. A cleared start or end hour is removed: the window then starts at 0:00, or ends at 24:00. A cleared category, severity floor or expiry date is removed too: the policy then applies to every category and every severity, with no expiry. Clearing one of these fields therefore widens the policy: it neutralizes more findings, and for good when the expiry is cleared. The justification is never removed: it is required, and clearing it makes the save fail with a message.

The package already ships **612 Custom Metadata records**, texts and reference tables. Each text is shipped in English and in French, hence two records per text: 156 remediation playbooks (312 `Remediation_Playbook` records), 14 root-cause rules (28 `Root_Cause_Rule`), 11 summary texts (22 `Posture_Narrative`), 7 insight texts (14 `Org_Insight`) and 4 recommendations (8 `Posture_Recommendation`). The other 75 `Posture_Narrative` records are the segments of the scan narrative, 43 in English and 32 in French: the 11 segments that only the emailed posture digest composes ship in English only. The reference tables add 76 `Remediation_Effort`, 62 `Data_Jurisdiction`, 7 `Parity_Rule`, 5 `Release_Calendar` and 3 `Attack_Path_Rule` records.

Sharing of ITSM Ticket lines (`Itsm_Ticket__c`). Ticket deduplication — the automatic creation pass as well as the **Create a ticket** buttons — reads **ITSM Ticket** lines with the user's permissions. The delivered sharing (public read/write) lets everyone see everyone's lines. If you make the object private, a user without “View All” on **ITSM Ticket** no longer sees the lines of others: the pass, which runs as the user who scheduled the job, may then create the ticket of a finding that is already tracked again, and an analyst does not see the pass's line. Keep the delivered sharing, or give “View All” on **ITSM Ticket** to that user and to those who click.

Wiring the `OS_SelfOrg` self-callout

9 features read your org **through its own API** rather than through an Apex query: this is the “self-callout”. It goes through the one Named Credential that the package ships, `OS_SelfOrg` (label **OrgGuardian self-org connection**), backed by the External Credential `OS_SelfOrg` (label **OS Self Org**). Both ship **unwired**: the address is a placeholder (`https://your-mydomain.my.salesforce.com`, which does not resolve) and the `SelfOrg` principal is empty. As long as they stay that way, every feature that depends on them fails closed: no token can be issued, nothing is read. The wiring is done **once** and serves every feature in the table below.

What the wiring opens. Each feature stays off until you switch it on in Administration ▸ Settings ▸ **Tooling API access** (list “What the wiring unlocks”), one at a time, or in **Feature activation**. The first one, **Tooling API discovery scanners**, opens every Tooling API read; the last four read through it, so they need it switched on too. Six of these features never follow a bulk action: **Tooling API discovery scanners**, the four that call your org's API themselves (Event Monitoring, limits supplement, silent

failures, record counting) and the certificate probe. The other three (org hardening, LWC and Aura linting, Agentforce agents) can be switched on by **Enable everything possible** when your plan includes them, but they read nothing until **Tooling API discovery scanners** is on.

What you get	Feature to switch on (screen label)	Plan	Condition besides the wiring
Health Check parity: of the 44 settings of the Standard baseline, 4 are recomputed through SOQL from installation; the **other 40**, the score and the verdict that the platform returns for each setting go through the self-callout. The same feature serves the discovery scans through the Tooling API (outbound callout posture, code quality)	**Tooling API discovery scanners** (`TOOLING_API_DISCOVERY`)	every org	—
Event Monitoring: reading the Event Log Files — EPT, slow SOQL, unhandled exceptions, concurrency and database CPU (the Operations Guide §8)	**Event Monitoring ELF fetch** (`EM_FETCH`)	paid	an Event Monitoring licence
Limits supplement: the additional limits of the REST `/limits` resource, on top of the org limits snapshot	**Org limits REST supplement** (`ORG_LIMITS_SUPPLEMENT`)	paid	—
Silent failures: outbound callouts that get an HTTP 200 whose body carries an error (rejected API key, `invalid_grant`); only the host is kept, never the full URL or the log content	**Silent callout failure detection (error inside an HTTP 200)** (`Callout_Silent_Failures`)	paid	active debug logs (trace flags); the product tells you when detection is blind
Record counting: the storage breakdown by object	**Storage breakdown by object** (`Monitoring_Record_Count`)	paid	—
Certificate probe: the expiry of your org's certificates	**Certificate and TLS expiry monitoring** (`Cert_Monitor`)	paid	**Tooling API discovery scanners** switched on too: the certificates are read through the Tooling API. The TLS probe of external endpoints goes through the Heroku option instead
Org hardening: password, session, clickjack and IP range settings read through the Tooling API	**Org hardening audit (passwords, sessions, IP ranges)** (`Org_Hardening_Audit`)	paid	**Tooling API discovery scanners** switched on too
LWC and Aura linting: unescaped output, manual DOM manipulation, dynamic code evaluation, iframes with a data-bound URL, read from the component source	**LWC and Aura XSS linter** (`CodeQuality_Declarative_Lint`)	paid	**Tooling API discovery scanners** switched on too

What you get	Feature to switch on (screen label)	Plan	Condition besides the wiring
Agentforce agents: the security posture of your AI agents, read from their metadata	**Agentforce AI agent security posture** (`Agentforce_Posture`)	paid	**Tooling API discovery scanners** switched on too

“Paid”: a paid plan, or the 30-day trial of the full plan.

The assistant (recommended): Administration ▸ Settings ▸ Tooling API access. It reduces the wiring to one manual step and checks the rest:

1. **Create an external client app.** The one step you do by hand, because Salesforce does not let a managed package create one (step 1 below). The assistant shows the callback URL to copy.
2. **Connect the credential.** In this version the assistant does **not** take the consumer key and secret: it shows the exact values to enter in Setup (your org's own URL for the named credential and for the token endpoint of the external credential, and the `SelfOrg` principal that receives the key and secret: steps 2 to 4 below), then reads the result back on its own every few seconds once you open Named Credentials from it. No URL is typed into OrgGuardian: both are derived from your org's domain, so the token can only go to your org.
3. **Run the test.** Two read-only requests to your own org: the first checks that the token is issued and the API answers, the second that the run-as user can read the Health Check. A Health Check that answers with no row counts as unreadable. The test passes only when the wiring works for the features you are about to turn on, including for the owner of the scheduled scan; otherwise the result names the missing piece (token refused, run-as user without View Setup and Configuration, owner of the scheduled scan without access to the principal, and so on).
4. **Turn on Tooling API discovery**, then the features of the list below it, one at a time. These buttons stay inactive until the test has passed on this screen, and OrgGuardian runs the test again just before each switch: if it fails, nothing is turned on. The list also counts the deep analyses that are active among those your plan includes, read in your org: no figure is sent anywhere.

The manual procedure below remains the reference: it is what step 2 of the assistant shows.

Before you start. You need your org's My Domain URL (Setup → **My Domain**, of the form `https://<your-domain>.my.salesforce.com`) and a System Administrator. In the steps below, replace `<your-domain>` with yours.

Step 1 — Create the external client app. Setup → **External Client App Manager** → new app:

- **Local** distribution;
- OAuth enabled, callback URL `https://<your-domain>.my.salesforce.com/services/oauth2/callback`;
- OAuth scope **Manage user data via APIs (api)**, and that one only;
- **Client Credentials Flow** enabled;
- in the app's OAuth policies, the flow's **Run As** user.

Every read made by the self-callout runs as that Run As user: what it cannot read, the self-callout does not read. For Health Check parity it needs **View Setup and Configuration**: this is what the read-refused finding says (below). Salesforce recommends a dedicated, API-only integration user for this flow. The

publisher validated the wiring with a System Administrator; the other rights that an integration user needs, feature by feature, have not been measured. Then note the app's **Consumer Key** and **Consumer Secret** (the app's OAuth settings).

Step 2 — Point the External Credential to your org's token endpoint. Setup → **Named Credentials** → **External Credentials** tab → **OS Self Org** → **Edit**: the URL of the token endpoint (the parameter that the Salesforce API names `AuthProviderUrl`) becomes `https://<your-domain>.my.salesforce.com/services/oauth2/token`. The shipped protocol (OAuth 2.0, client credentials) cannot be changed. Do not add a **Scope** parameter: the Salesforce token endpoint rejects it; the scope comes from the client app (step 1).

Step 3 — Enter the credentials on the principal. Same page, principals section → **SelfOrg** principal → authentication parameters: **Client ID** = the Consumer Key, **Client Secret** = the Consumer Secret from step 1. These values stay in your org's credential store: no package field holds them, and the package neither reads nor logs them; the assistant never asks for them. If lost, regenerate them in the client app, then enter them again here: no other place holds them.

Step 4 — Point the Named Credential to your org. Setup → **Named Credentials** → **OrgGuardian self-org connection** → **Edit**: URL `https://<your-domain>.my.salesforce.com`. The Named Credential ships in subscriber mode: its URL is edited in place, there is nothing to create. Do **not** create a local Named Credential with the same name: the package calls the one it ships, `orgguardian__OS_SelfOrg`.

Step 5 — Grant access to the principal. The call runs as the user whose transaction makes it: for scheduled scans, the user who scheduled the jobs; for a scan started by hand, the user who starts it. That user needs access to the `SelfOrg` principal of the External Credential (`orgguardian__OS_SelfOrg-SelfOrg`): among the shipped permission sets, only **`OrgGuardian_Admin`** grants it. Without it, the call answers “We couldn't access the credential(s)”.

Step 6 — Switch on the features you want, one at a time, in **Tooling API access** or in **Feature activation** (table above). The **Connect Tooling API access** step of the setup tour on the **Home** screen opens the assistant.

Checking.

- The assistant's **Run the test** checks the wiring end to end before you turn anything on, and turning a feature on from the assistant runs it again first.
- The **Connect Tooling API access** step of the setup tour is ticked when the Named Credential points at your org and **Tooling API discovery scanners** is on. OrgGuardian reads that URL through the platform's Connect API (the Named Credential object does not show it): if that read is refused, the step says it cannot verify rather than tick itself.
- As long as the Health Check is not read, a finding of severity **Info**, with target “Salesforce Health Check” (source `Security:HealthCheck:Access` in the finding's details), says why in its message: Tooling API discovery is off; the Named Credential did not answer (not pointed to your My Domain, or not authorized: steps 2 to 5); the Tooling API refused the query (the Run As user lacks View Setup and Configuration); or the read did not complete in this scan. It disappears at the first complete read. Rather than wait for the hourly scan, start one in anonymous Apex, as a user who

holds `OrgGuardian_Admin`: `System.debug(orgguardian.OrgGuardianAdmin.runScanNow(false));` (the Installation Guide §5.3). The setup tour's **Run scan now** button does not help here: it is offered only until the first scan completes, which the bootstrap has usually already run.

- Direct check, in anonymous Apex, as a user who holds `OrgGuardian_Admin` (a read; nothing is written):

```
HttpRequest req = new HttpRequest();
req.setEndpoint('callout:orgguardian__OS_SelfOrg/services/data/v67.0/limits');
req.setMethod('GET');
System.debug(new Http().send(req).getStatusCode()); // expected: 200
```

What remains true after the wiring. The self-callout calls one host only, your own org's My Domain, at the addresses you set in Steps 2 and 4: no third-party host receives it. It still counts among the outbound features, because these are HTTP requests, and each one uses one request of your org's daily API limit. There is no self-callout by session token (the Security FAQ §3).

What was measured, and what was not. The publisher wired the self-callout on 2026-08-05 on an org where an early version of the package was installed: the `/limits` call above returned 200. Steps 1 to 3 were carried out there through the platform APIs, not through the screens described here, whose page and field names have not been checked one by one; access to the principal through `OrgGuardian_Admin` (step 5) entered the package after that measurement. The procedure has not been replayed since: please report any difference to us. The assistant does not write the credentials in this version: a write by the package itself has not been measured on an installed package, so it ships turned off and the assistant shows this procedure instead.

Instrumenting your own callouts (`OrgGuardianCallout.execute`)

The package exposes a `global` method that **your** Apex can call to route its own HTTP callouts through it, so that OrgGuardian records their telemetry. It is optional: nothing in the product calls it for you, and without it the **Callout instrumentation** feature has nothing to measure.

Signature (`global with sharing` class, `orgguardian` namespace):

```
global static HttpResponse execute(HttpRequest req)
```

It sends **your** request as is and returns the response as is: it changes neither the address, the method, the headers nor the body, and adds no call of its own — no retry, no circuit breaker, no fallback.

Example, in a class of yours:

```
public with sharing class BillingClient {
    public static HttpResponse sendInvoice(String jsonBody) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:Billing_API/v1/invoices'); // your Named Credential
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/json');
        req.setBody(jsonBody);
        // instead of: return new Http().send(req);
        return orgguardian.OrgGuardianCallout.execute(req);
    }
}
```

What the call does, depending on the Callout instrumentation feature (`Callout_Instrumentation`).

- **Off** (the default): plain pass-through. The request is sent, the response comes back, nothing is written: the behaviour is that of `new Http().send(req)`.
- **On** (paid plan; Administration ▸ Settings ▸ **Feature activation**, individually: it counts among the outbound features and never follows a bulk action): after each call, the observation leaves as a **platform event**, **Callout Log Event** (`Callout_Log_Event__e`), published immediately, and the package's trigger saves it as one **Callout Log** row (`Callout_Log__c`) **outside your transaction**, a few seconds later (under 15 s in both of the publisher's measurements of 2026-10-06). Nothing is written in your transaction. The row carries the Apex entry point `Class.method` that called `execute` (read from the stack trace, at most 120 characters, `UnknownEntryPoint` otherwise), the destination host only (`https://host`, or `callout:CredentialName` for a Named Credential — never the path, the query string or a token), the HTTP status, its class (`2xx` to `5xx`, `ERR` for a transport failure), the response time in milliseconds, a normalised error type and the timestamp. Headers and body are neither read nor kept. It also carries the **Id** of the user whose transaction made the call (field **Caller Id**, `Caller_Id__c`: the Id alone, no name, no email), taken from the published event: since the row is created by the trigger, its **Created By** field names the trigger's user, the same for every row, not the caller. Its **owner** (field **Owner**), however, is the caller: this is what lets the retention purge delete it (limit 4).
- **Transport failure** (timeout, DNS, TLS): the failure is classified and logged (feature on), then **the original exception is re-thrown**: your error handling runs as it would without the wrapper. An error while publishing or writing the telemetry, on the other hand, is swallowed: it never breaks your call.

What the product derives from it.

- **Outbound callouts** screen, **Failed callouts** card: your calls by Apex entry point and by host — totals, failures, timeouts, 4xx, 5xx, failure rate, average time — over the last 24 hours (at most 10,000 rows). With the feature off, the card stays locked.
- **Endpoint dependency graph** (feature **Endpoint dependency graph**, paid plan, on by default): at each scan of the Callouts pack, a “called by” edge links each Apex entry point to the host or Named Credential it calls, from the 5,000 most recent Callout Log rows. The graph's second source, the endpoint registry (feature **Endpoint registry**, every org, off by default), links the

endpoint to the Named Credential or remote site that carries it. The **Config impact** screen, integration axis, then answers “if this endpoint or this Named Credential goes down, what breaks?”. An edge that no scan has seen again for 30 days leaves the screen.

- **Incidents**: an incident's impact estimate counts the failed calls (4xx, 5xx, transport failures) during its window, and the distinct users whose calls failed (field **Caller Id**). A row without that field is not counted; with none, the number of affected users falls back to the number of distinct targets of the incident's findings. The count is made when the incident is closed, with the rights of the user who closes it or runs the scan: without read access to **Caller Id**, it keeps the number of failed calls and falls back for the users only. Among the shipped permission sets, `OrgGuardian_Admin`, `OrgGuardian_Analyst` and `OrgGuardian_Auditor` grant that read access, `OrgGuardian_Viewer` does not (the Id of an observed person).

Limits.

1. **Several instrumented calls in one transaction.** With the feature on, `execute` writes nothing in your transaction: it publishes a “Publish Immediately” event. Salesforce refuses a callout that follows a pending operation — a write, asynchronous Apex, sending an email (Apex Developer Guide, “Callout Limits and Limitations”) —, and an event published immediately leaves at the publish call, without waiting for the end of the transaction (Platform Events Developer Guide, “Platform Event Fields”; both read on 2026-10-06). Measured by the publisher on 2026-10-06 on its development org, feature on: two `execute` calls in one transaction both got their response, with no DML statement in the transaction, and left two Callout Log rows. Not yet measured on an installed package.
2. **One right for the user who runs your code: create on the Callout Log Event event.** Salesforce refuses to publish an event for a user who lacks create access on that event, even from the package's code (measured on 2026-10-06). That user needs **no** right on the Callout Log object or its fields. Among the shipped permission sets, only `OrgGuardian_Admin` grants it; for an integration user, add read and create on the **Callout Log Event** event to a permission set of your own (a platform event carries no field-level security). Without that right, the telemetry is dropped **before** publishing, and your calls run exactly as with the feature off. The row itself is written by the package's trigger, in user mode, as Salesforce's **Automated Process** system user, under which event triggers run by default. Measured on the publisher's development org: the write succeeds. Not measured on an installed package: if it were refused there, your calls would not be affected, and the loss would be recorded in the **System Log** (category `CALLOUT_LOG_EVENT`) whenever that log can be written. The trigger assigns the row to the user who published the event, not to Automated Process, which Salesforce would otherwise make the owner (Platform Events Developer Guide, “Considerations for Publishing and Subscribing to Platform Events with Apex and APIs”, read on 2026-10-06). Measured on 2026-10-06 on the publisher's development org, including for a user who holds only create on the event and no right on Callout Log. If Salesforce refuses that owner (for instance a user deactivated between the call and the save), the row is saved in the name of Automated Process rather than lost.
3. **A Named Credential of your org.** The call is made from the package's code, in the `orgguardian` namespace. A Named Credential of your org (`callout:Your_NC/…`) can be used there only if it allows that namespace (parameter `AllowedManagedPackageNamespaces`, **Allowed Namespaces**, set to `orgguardian`). This is the platform rule for a managed package's code, measured by the

publisher on a call from the package to a Named Credential of a subscriber org (no successful call without that permission, one with it); it has not been measured on a call made through `execute`. A plain address (`https://…`) remains subject to your remote site settings.

4. **Cost.** One published event per instrumented call, and no DML statement in your transaction. The event counts against a separate budget of 150 “Publish Immediately” publish calls per transaction, not against the 150 DML statements: once that budget is spent, the telemetry is dropped, never your transaction. It also counts against your org's hourly event publishing allocation (250,000 per hour in Enterprise, Unlimited and Performance editions). Before publishing, the package reads the allocation left that the platform reports, and drops the telemetry when it no longer covers it. This is a **best-effort** guard: the reported counter lags. Measured on 2026-10-06 on a development org: three Apex publishes counted within their own transaction, then 0 read in a later transaction 28 s afterwards and again 53 min afterwards, the *Limits* API still returning 250,000 left. Close to the exact limit, a publish can therefore go out and be refused; whether a refusal for exceeding the allocation (`LIMIT_EXCEEDED`) makes the next call of the transaction fail is not measured (a refusal for a missing permission does). The event does not count against the delivery allocation, which an Apex trigger does not consume (Platform Events Developer Guide, “Publish Event Messages with Apex” and “Default Platform Event Allocations for Event Publishing and Delivery”, read on 2026-10-06). Add the storage of the Callout Log rows, purged according to the log retention (`Retention_Log_Days`, 30 days by default, 7 on a small org, **General settings** pane). The purge runs every night in user mode, as the user who bootstrapped the scheduled jobs (installation guide, §5.2 and §5.3), and the **Public Read/Write** sharing of Callout Log grants editing a row, not deleting it. It therefore deletes the rows that user owns, all of them if that user holds “Modify All Data”, and, depending on the role hierarchy, those of the user's subordinates (platform rule, not measured). Measured on 2026-10-06 on the publisher's development org: a row owned by another user survives the purge; the row of the caller who bootstrapped the jobs is deleted. If your instrumented calls run as an integration user who is neither that user nor a subordinate, and that user does not hold “Modify All Data”, their rows stay beyond the retention; every night, a `RETENTION_PURGE` warning in the **System Log** counts the rows that were not deleted, and a Low finding (source `OrgGuardian:RetentionRefused`, target `Callout_Log__c`) names their number and the cause; it closes on its own at the first purge of Callout Log with no refusal.
5. **A fixed contract.** This is the permanent API of a managed package: once a version is released, the `execute(HttpRequest)` → `HttpResponse` signature can no longer change.
6. **Immediate publishing, at-least-once delivery.** The event leaves at the time of the call, even if your transaction fails afterwards: a call made in a rolled-back transaction keeps its row. Salesforce delivers an event at least once: a duplicate is rare but possible (Platform Events Developer Guide, “Considerations for Defining and Publishing Platform Events”, read on 2026-10-06).

---

4. What configuration never touches

No OrgGuardian setting writes:

- standard Salesforce metadata;
- a metadata type created by the customer or by another package;

- business data;
- code;
- a permission.

This is the formal statement of the publisher's Metadata API notice, available on request. You can check it yourself: every type that an editor writes carries the namespace prefix `orgguardian__` (Setup → Custom Metadata Types).

---

5. Where to go next

- Who is allowed to open these screens: Permissions and Personas.
- What the configuration produces day to day: Operations Guide.
- A screen that does not behave as described here: Troubleshooting Guide.
- What the product reads, writes and sends: Security FAQ.
- The Metadata API notice and the activation of the optional features (Event Monitoring, multi-org aggregation, SIEM export, ITSM integration…): separate publisher documents, available on request.
- Overall architecture: the product's architecture note, available on request.