Security FAQ

*Audience: CISO, DPO or security architect assessing the installation of OrgGuardian in a Salesforce org. Each answer describes the package as shipped; where an answer can be checked in your own org, the Setup path or the query to run is given. The version installed in your org can be read in **Setup → Installed Packages**.*

> **This document set is self-contained.** Where a document from the publisher's compliance dossier supplements an answer, it is marked “available on request”: these documents are not part of this set, and are provided on request for your security assessment. Object, field and custom metadata type API names are given without the `orgguardian__` namespace prefix that they carry in your org; the queries below carry it.

---

0. What we do not claim

The product does not remediate. It detects and advises. It modifies no security configuration, disables nothing, revokes no access. This *read-only / detect+advise* posture is an architectural constraint, not a runtime option.

---

1. What exactly does the product read?

Five families of sources, all read-only.

Family	Example sources	Note
Setup objects	`SetupAuditTrail`, `PermissionSet`, `PermissionSetAssignment`, `User`, `CronTrigger`, `AsyncApexJob`, `EventLogFile`	Each read is handled on its own: a refusal **degrades** the collection; it does not interrupt the scan. The refusal is reported by an explicit finding, with exceptions (below). A refused `AsyncApexJob` read follows this rule. It produces a finding that names it: the checks on asynchronous jobs (failures, queue depth, run times) are skipped, the unhandled-exception count is recorded without a value rather than as a partial count, and the findings these checks had already opened, unhandled-exception findings included, stay open (Troubleshooting Guide §4 and §9)
Org **metadata**	global describe, configuration dependencies, site settings, installed packages	Read-only
Objects **of the package itself**	`Finding__c`, `Org_Metric_Current__c`, `Scan_Run__c`, `Incident__c`…	The records that the product itself wrote
Public file shares	`ContentDistribution` (the configuration of the link: password required or not, expiry, number of views, creation date), and the **title** `ContentVersion.Title` of the file shared through that link	The only read that touches a content object of the subscriber. Neither the content of the file, nor the public URL, nor the password, nor the author is read. **The title is never kept**: it is reduced to a single boolean — whether the title carries a sensitive marker — then discarded, and appears in no finding, no metric, no log
Sharing tables (feature **Public site posture and site-to-guest inventory**, paid plan, on by default)	`AccountShare`, `ContactShare`, `<Object>__Share`…: one record count per object that the guest user of an active site can read, limited to the guest sharing causes (`GuestRule`, `GuestParentImplicit`, `GuestPersonImplicit`) and to the guest users of those sites	A number, never a row: no record Id, no user Id, no field value. At most 10 counts per scan

The product does not read your business data. It reads no account, contact or opportunity records, nor records of the subscriber's custom objects. The only measurement that touches their sharing tables is a count, described above. Its subject of study is the org's **posture** — permissions, configuration, technical logs, health of the integrations.

The only exception, and why it exists. A public distribution link (`ContentDistribution`, a file's public share link) opens a file in your org **without login, without profile and without sharing rule**, and it survives the deactivation of its author: it is a direct exfiltration channel. A detector that did not look at it would leave a large hole outside its scope. To prioritise links, the product needs to know whether a given

link points to a file that is likely to be sensitive — hence reading the **title**, and the title alone. The product reads neither the `DistributionPublicUrl` nor the `Password` field of the link, and the title serves only to classify the link before being discarded. The corresponding code can be shown on request.

What you can check for yourself: open any `Security:PublicLink:*` finding — it carries only the opaque 18-character identifier of the distribution, configuration booleans and counts. To list them:

```
SELECT orgguardian__Target__c, orgguardian__Message__c
FROM orgguardian__Finding__c
WHERE orgguardian__Dedupe_Key__c LIKE 'Security:PublicLink:%'
```

If the user running the scan has no access to these objects, the product **says so** through a finding that names the cause — it does not stay silent, and it never publishes an all-clear on risky public links that it has not measured.

Not every refused setup read is reported in this way: some degrade without producing any finding — the bootstrap state read from `CronTrigger`, for one, which the Home screen then reports as unverified, in the banner “Bootstrap state cannot be verified with your profile” (Troubleshooting Guide §4 and §5). For the checks that such a read feeds, the absence of a finding proves nothing.

What “MFA” means in a finding. Two detectors speak of it, and neither sees the second factor that an identity provider requires:

- the **MFA gap** (`Security:MfaGap`) reports each active human account that no profile or permission set covers with the “Multi-Factor Authentication for User Interface Logins” permission: one finding per account up to 50 accounts, a single summary finding beyond; the published rate, `Security:MfaCoverage` (“MFA Coverage”), relates the covered accounts to all the active human accounts. The org setting “Require multi-factor authentication (MFA) for all direct UI logins to your Salesforce org” (Identity Verification) is not read: it protects as well, but does not close the finding;
- the **toxic combination** (`Security:BlastRadius`) targets an active account that holds data access (“Modify All Data” or “View All Data”) and “API Enabled”. **By default, it is assessed without MFA**, even under an administrator running the analysis: Salesforce's registry of verification methods (`TwoFactorMethodsInfo`) can be read only with the “Manage MFA in API” permission (object reference, “Special Access Rules”: “You need the Manage MFA in API user permission to access this object”, read on 2026-10-03), which no shipped set carries, and which the standard profiles do not carry either: `PermissionsManageTwoFactor` is false on the System Administrator, Standard User and Minimum Access profiles (measured on 2026-10-03 on an org in its default installation state; “Manage MFA in User Interface”, `PermissionsDelegatedTwoFactor`, true on System Administrator, does not open that registry). The combination then opens in `High`, and its sentence says that MFA is not assessed. Granting “Manage MFA in API” to the scan account, which runs the scheduled analyses, lets MFA be assessed: the combination becomes `Critical` when the registry, read, holds **no MFA method** for the account (Salesforce Authenticator, an authenticator app, a security key or a built-in authenticator; a verified email address, a mobile number or a temporary code, which the registry also records, does not count), and closes if it holds one. **Warning**: the same permission also opens the writing of users' MFA records through the API (the `TwoFactorInfo` object: create, update, delete); weigh it before granting it, then grant it to the scan account only, never to every user who clicks “Run scan now”: that button runs as the user who

clicks, a run launched without the permission assesses the combination without MFA, and the next scheduled scan, run by the scan account, reads the registry. A registry read that falters (refused, failed, no row returned, probe skipped or cut by the row budget) closes, reopens or changes the key of no combination of an account it did not read: an open combination goes back to `High`, a closed one does not reopen, and the change between `High` and `Critical` is never counted as a regression; a read cut short assesses the accounts it read (`Critical` with no method, combination closed with one). If the analysis cannot re-read the combinations of those accounts in full, it keeps only the ones already open, opens none, and the notice says so. The population metric is recorded without a value while the registry is not read in full, and a read notice gives the cause (the permission, the probable cause of a refusal; another failure or an empty registry is said otherwise). A combination closes when a successful read of the assignments shows that its account lost the data access or the API. The object reference caps an API query at 2,500 rows, a cap that is not measured in Apex: a read that returns 2,500 rows or more is therefore treated as cut short, with the effects described above for the accounts it did not read.

An MFA required by your identity provider (SAML or OpenID Connect single sign-on) is **invisible to both, by design**: its policy lives outside the reach of Apex, and the product never infers coverage from a link to a federation alone, which says neither that single sign-on is active nor that it requires a second factor. Instead, next to `Security:MfaCoverage`, it publishes the number of active human accounts analyzed whose `FederationIdentifier` field is filled in (`Security:IdentityFederated`, "Federated Identity Accounts"): if your MFA lives in the IdP, those two figures let you decide. That counter is not published, rather than published at zero, when field-level security denies `FederationIdentifier` to the user who runs the scan.

---

2. What does it write, and what does it never write?

It never writes to your business data, nor to your security configuration: no profile, no permission set, no sharing rule, no org setting, no business standard-object record.

What it writes, exhaustively:

1. **Its own objects** — 20 custom objects (the fleet inbox and the setting-level detail of the signed baselines included) and 9 custom settings (`__c`; `OrgGuardian_Compliance_Map__c`, item 2, among them), 7 Big Objects (`__b`) and 2 platform events (`__e`; one of them, `Callout_Log_Event__e`, carries the telemetry of `OrgGuardianCallout` to the trigger that saves it), all in the `orgguardian` namespace (full inventory in the Glossary).
2. **One file in Files**, and only one, when an administrator replaces the shipped compliance mapping table: a `ContentVersion` titled `OrgGuardianComplianceMap-override`. The file 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 the imported version is **recorded in a protected setting** (`OrgGuardian_Compliance_Map__c`) that only the package code reads and writes. Only that version applies: a file with the same title, private or shared by someone else, is neither applied nor deleted. Reverting to the publisher's reference clears the record and touches only the recorded document: it deletes it when the administrator who reverts owns it, and otherwise withdraws its share with the organization.

3. **An installation marker** in the custom setting `OrgGuardian_Install__c` (timestamp and version numbers — see §5).
4. **Records of its own Custom Metadata Types**, only when an administrator saves from a no-code editor or imports a rule pack: they are deployed through the Metadata API, never by DML (see §1 of the Configuration Guide).
5. **Its scheduled jobs**, all named with the `OrgGuardian` prefix (Setup → Scheduled Jobs).
6. **Chatter posts**, only once an administrator has enabled the Chatter alert channel.

Every DML write among these runs in Salesforce **user mode** — with the object permissions, field-level security and sharing of the running user — apart from the two exceptions named and bounded in §5.

---

3. What leaves the org, to where, and under what condition?

Short answer: by default, no outbound call. The package has **22 outbound features**, in four families: HTTP alert channels (5, reading Slack reactions included), exports and integrations (4), external probes (7) and calls to your own org's API (6). All of them are off at installation and are enabled one by one — a bulk action never enables any of them. By default, alerts are delivered inside your org (a platform event and an in-app notification): email alerts start only once an administrator activates an email channel.

The egress surface and credentials are also the subject of a dedicated document in the publisher's compliance dossier, available on request.

One platform channel is not a package call: error notifications to the publisher. When code of the package raises an unhandled Apex exception in your org, or when an installation or upgrade fails, **Salesforce itself** (not the package) emails the publisher's designated error-notification user, a standard mechanism of every managed package that you do not configure. Such an email carries your org Id, the Id of the user who ran the code, the exception type and message, and the stack trace; an exception message can quote a value the code was handling. The package makes no call for this and stores nothing about it.

The outbound surface in five measurements.

Measurement	Value	Where to check it
Package classes that open an HTTP connection	**15**	code-level count, available on request
HTTP send sites in the package code	**22**	code-level count, available on request
Named Credentials shipped	**1** — `OS_SelfOrg`	Setup → Named Credentials → **OrgGuardian self-org connection**
Hard-coded external host	**1**, declared: `https://viewer.orgguardian.com`, the default endpoint of the Heroku option	the **Heroku option** pane of the Settings tab shows the endpoint in force (see §6)
Callouts that use a user's session ID	**0** — the package mentions the session ID only in the text of its own code-quality advice, which warns against the practice	code-level count, available on request

The 22 send sites and the 22 outbound features are two separate counts, equal by coincidence: one send site can serve several features, and one feature can have none of its own (the automatic ITSM ticket creation, for one, sends through the ITSM integration).

The three cumulative locks.

1. **Switch.** Each egress path is a feature switch, and **all** of them are off by default: an installation left at its defaults therefore makes no outbound call. The **Feature activation** screen shows the state of each switch.
2. **Plan.** Each feature also carries a licence plan. An org whose plan has not been provisioned by the publisher resolves to the lowest plan, `FREE`, and fails every paid-plan check — except during the 30-day trial of the full plan, which starts when an administrator first opens an OrgGuardian screen. A sufficient plan **opens no flow on its own**: the two locks are independent.
3. **Fail-closed.** Any uncertainty closes the path: no endpoint, no consent where the path asks for one, or a transaction in which the platform would refuse the call ⇒ nothing is sent, never a partial send. A path that signs what it sends — the SIEM export, the ITSM integration, the Heroku publication, the generic webhook alert channel — sends nothing without its key. The webhook channel also timestamps every send, and sends nothing without its Named Credential and a recorded consent: see the Webhook Receiver Guide (available on request). The connection test and the activation exchange of the Heroku option are not signed, the exchange because it precedes its key (§6). Slack, Teams and PagerDuty do not sign either: their destination verifies no signature, and authenticates the send by the address of its Named Credential or by the integration key that you provide; nor do the reads of the fleet, authenticated by the Named Credential of the linked org. *(Five classes of the package compute an HMAC: `HerokuPublisher`, `ItsmTicketService`, `SiemExportService`, `WebhookAlertChannel` and `SubjectPseudonym`. `SubjectPseudonym` computes the subject pseudonym with a key held in a protected setting of your org, which only the package code reads: it is not a send signature.)*

Calls to your own org are outbound calls too, and they are off as well. They go through the one Named Credential that the package ships, `OS_SelfOrg`, meant to point to your org's own My Domain. It ships with a non-resolving placeholder address, which your administrator replaces (Setup → Named Credentials → **OrgGuardian self-org connection** → Edit) before enabling any of these features — until then, such a call fails closed. There is no self-callout by session token: calls through this credential authenticate with the identity that your administrator configures on it (an OAuth client-credentials flow), and among the shipped permission sets only `OrgGuardian_Admin` grants the use of that credential. The full procedure, and what each feature opens: the Configuration Guide §3, "Wiring the `OS_SelfOrg` self-callout".

The Tooling API access assistant and the client secret. Administration ▸ Settings ▸ **Tooling API access** guides your administrator through this wiring. It never asks for the consumer key or secret: your administrator enters them in Setup, on the `SelfOrg` principal of the external credential, and the package neither reads nor logs them. The two addresses it shows to copy are derived from your org's own domain, so the token can only be requested from, and used against, your org. Its end-to-end test sends two read-only requests to your own org, on your administrator's click only, and turning a feature on from the assistant runs that test again first: if it fails, nothing is turned on. No scheduled job, batch or installation step ever runs the test. The assistant requires the OrgGuardian administration authority. The activation signal it displays is read in your org and sent nowhere.

What goes over the wire, when a channel is explicitly enabled. Before text leaves through an HTTP alert channel, the SIEM export or the ITSM integration, OrgGuardian replaces any Salesforce user identifier (`005…`) in it with `user:` followed by the first eight hexadecimal characters of an unkeyed SHA-256 digest of the 18-character Id. This is a stable pseudonym (not anonymised): anyone who holds the list of your org's user Ids can recompute the mapping. For the signed webhook, the substitution takes place **before** signing, so the HMAC digest covers the bytes actually sent. The webhook body carries neither the finding's sentence nor its target field (`Target__c`): twelve keys, among them `source`, the only free value, to which the substitution applies. For some checks, `source` names the watched item (a host, a metric, a limit, a setting, a permission, the ITSM tool…), which, for several of them, is the target of the finding itself; never a user identifier. Details: the Webhook Receiver Guide (available on request), §10.

What comes back from a third party: Slack reactions. When your administrator consents to acknowledgement by reaction (`Chat_Reactions` feature, off by default, paid plan, `ENTERPRISE`, never opened by a bulk action), the org posts its Slack alerts through Slack's Web API (`chat.postMessage`), through a Named Credential of its own for the bot, instead of the incoming webhook, whose Named Credential does not change; then it reads, every hour, the reactions on these messages (`reactions.get`: at most 25 reads per pass, messages less than 72 hours old, never a message posted before the consent). From the post, it keeps the message's timestamp and channel ID. Slack's answer about the reactions carries their **names** and the identifiers of their authors (the package always asks for the full list, `full=true`): the names decide the effect; the identifiers are used only if a list of allowed members is set, compared to that list in memory, then forgotten. No member identifier returned by Slack is kept, written or logged, and no Slack member is matched to a Salesforce user: the acknowledgement is recorded with no author, and so is its activity line. The bot token, which authenticates both calls, lives in the External Credential of the bot's Named Credential, never in a package field. No entry point is opened: the org always makes the call (the Operations Guide, "Acknowledging from Slack with a reaction").

A refused send leaves a trace. When an HTTP alert channel refuses to send — personal data detected in the payload, an endpoint that is not HTTPS, or the per-transaction cap reached — it writes a `System_Log__c` row in the dedicated `OUTBOUND_BLOCKED` category, with a cause token. A ticket that the ITSM integration refuses to send is reported on screen instead, in a "Ticket not created" message that gives the reason. An administrator can list the logged refusals:

```
SELECT orgguardian__Logged_At__c, orgguardian__Message__c
FROM orgguardian__System_Log__c
WHERE orgguardian__Category__c = 'OUTBOUND_BLOCKED'
```

---

4. What inbound surfaces does the package expose?

Two Apex REST resources; the package exposes no other web service:

Resource	Path in your org
Incidents	`/services/apexrest/orgguardian/orgguardian/incidents/*`
Posture status	`/services/apexrest/orgguardian/orgguardian/status/*`

The first `orgguardian` in each path is the namespace prefix that Salesforce adds to the REST paths of a managed package. The status resource answers from installation (feature **REST API**, on by default), except `/deploy-check`, which waits for its own feature, **Deployment advisor (CI check)**, off by default, and `/parity-fingerprint`, which waits for two, **Parity fingerprint API** and **Sandbox and production parity**, both off by default — the first is never switched on by a bulk action. By default, `/findings` withholds the target and the sentence of each finding, which can name a user account: they are served only once an administrator enables **Finding details in the REST API**, off by default and never switched on by a bulk action either. The incidents resource answers only once the **Incident management** feature, off by default, is enabled.

The incidents resource also accepts one write: a `PATCH` that moves an incident's status (Open → Investigating → Resolved) or records its root cause. It has its own feature switch, **REST write on incidents (PATCH)**, off by default, and it requires edit access to `Incident__c`, which among the shipped permission sets only `OrgGuardian_Admin` grants.

The **Chat acknowledgement link** feature opens no inbound surface: it only controls the "Acknowledge in Salesforce" link in Slack and Teams alerts, which opens the finding's page in your org, under your own session.

Both resources return an `OrgGuardian-Schema-Version` header on every response, `403` and `404` responses included. Its current value is `1`. It carries the version of the **response contract**, not that of the package: it is incremented only if the shape of a payload changes in a **non-additive** way — a field removed, renamed, or whose type changes. Adding a field does not increment it.

Why a header rather than a `/v1/` segment in the URL: what breaks an integrator is the **shape of the payload**, not the path — and two of the seven `/status` responses are JSON **lists** (`/findings` and `/metrics`), which would have had to be wrapped in an object to hold a number, that is, to cause exactly

the break that this marker exists to prevent. The seven sub-routes: the summary (bare `/status`), `/findings`, `/metrics`, `/sla`, `/health-summary`, `/deploy-check` and `/parity-fingerprint`.

These are **standard** Apex REST resources: the platform requires an **authenticated Salesforce session**. There is no unauthenticated public endpoint in the package.

The caller receives nothing that a direct query under the same identity would not already give them: both resources read in user mode and apply the caller's sharing.

The package also exposes **5** global Apex classes, the permanent API of a managed package: the two REST resources above; an Agentforce action that queries the posture or snoozes or acknowledges a finding (feature **Agentforce actions**, off by default); an optional helper, `OrgGuardianCallout`, through which your own Apex can send its own HTTP callouts, whose telemetry OrgGuardian records once **Callout instrumentation** is enabled (how to use it, and its limits: the Configuration Guide §3, “Instrumenting your own callouts”); and an administration facade, callable from anonymous Apex, which schedules the product's jobs, starts a scan, or queues a publication of the Heroku option that the conditions of §6 still govern (see the Troubleshooting Guide).

---

5. The “user mode” posture, and its two exceptions

The rule. It applies without exception to **everything that the product reads in your org**:

- Every SOQL / SOSL query runs in **user mode**, which enforces the running user's object permissions, field-level security and sharing — never in system mode, never without an explicit mode.
- Every write to a record runs in user mode likewise.
- Every part of the package that touches data enforces the sharing of the running user.

Exactly two exceptions, and both are writes. We count them here rather than let you discover one. The first stems from the identity that Salesforce imposes on the install script; the second stems from an API that Salesforce does not provide. **Neither of them reads data that belongs to you**: the first touches only package objects; the second writes history rows into package objects. Some of those rows are per user: the login habits of privileged users in `Login_Baseline__b`, under the feature **Login anomaly detection (country, impossible travel, off-hours)**, and the effective-rights register in `Config_Snapshot__b`, under the feature **Permission-sprawl detection** — both features of the paid plan, off by default (see §2 of Permissions and Personas). In other words, **everything that the product reads in your org is read in user mode, without any exception**; the two exceptions live elsewhere, on the write side.

Direct consequence: **OrgGuardian never sees more than the user it runs as.** A read-only user who opens a screen cannot, through the product, reach data that their profile denies them. The one identity that you choose separately is the one configured on the **OS Self Org** external credential, behind the **OrgGuardian self-org connection**: the calls to your own org's API (§3), off by default, see what that identity sees. This is also what explains the degradations described in the Troubleshooting Guide: the product prefers an incomplete result — not always flagged, see §1 — to a complete result obtained in system mode.

The scan account needs four read permissions that the package cannot grant. A packaged permission set carries no system permission, and none of the five shipped sets does. Scheduled analyses run as the user who armed them; that scan account needs View Setup and Configuration, View Roles and Role Hierarchy, View All Users and View All Profiles, all read-only, which a system administrator already holds through the profile. At the start of every analysis, the product reads the effective permissions of the account that runs it: each missing one is named by a `High` read notice (`OrgGuardian:ScanRights:<API name>`), and the controls that depend on it are neither credited with a pass nor closed as fixed (Permissions and Personas, §2, “The scan account: four read rights that you grant”).

Exception no. 1 — the installation trace, named and bounded

The install script writes the installation marker in system mode. The full reasoning is set out in a dedicated document of the publisher's compliance dossier, available on request.

The scope, bounded along seven dimensions:

- **When** — only while the install or upgrade script runs. Never during operation.
- **What** — two writes: the durable marker, and a log line attempted on a best-effort basis.
- **Where** — two stores belonging to the package: the List custom setting `OrgGuardian_Install__c` (**Public** visibility, so that a subscriber administrator can read the trace: Setup → Custom Settings → **OrgGuardian Installation** → **Manage**) and the logging object `System_Log__c`.
- **Content** — fixed text and what the platform itself provides: date and time, installed version number, upgrade flag, previous version. No user input, no concatenation of org data.
- **Subscriber data read** — none.
- **Subscriber data written** — none.
- **Outbound call** — none.

Volume: one record per store and per installation.

Why user mode is technically impossible here. A post-install script runs as a package system user that **has neither a profile nor a permission set** and cannot even be queried in the `User` object. In user mode, every read and every write of that user is refused: without this exception, an installation would leave no trace at all — not even the log line meant to report it.

Exception no. 2 — the history snapshots, imposed by the platform

The history snapshots are written to, and purged from, the package's Big Objects through the platform's immediate insert and delete operations for Big Objects.

This one is not a choice: it is the absence of an API. These two operations accept **no** access mode — user mode **does not exist** for Big Object writes in Apex. No write of this kind, in any Salesforce product, can run in user mode. We could therefore neither enforce it nor forgo it without forgoing the history itself.

What bounds it: these objects **belong to the package**, never to you; they receive only rows that the product derives — counters, measurements, states and fingerprints, plus the per-user rows named above — never a copy of a business record; the queries that feed them run in user mode; and when the

platform would refuse the write, the product skips it rather than forcing it — a failed snapshot never interrupts a scan. These two operations are used nowhere else in the package.

No other operation of the package escapes user mode apart from these two.

---

6. The Heroku option: what does it change?

It is off by default, and publishing requires four conditions, all of them cumulative.

Until an administrator acts on the **Heroku option** screen, this path makes no outbound call, and the product works in full without it. Besides the publication itself, described below, acting on that screen can send three kinds of call to the collector, at the endpoint that the screen shows (by default `https://viewer.orgguardian.com`), and none of them is signed:

- the **connection test**, when an administrator clicks **Test the connection**, sent only once conditions 1 to 3 below are met: an unauthenticated `GET /healthz`, with no body;
- the **activation request**, sent by the screen's **Activation assistant**: your org Id and the SHA-256 fingerprint of a claim secret that your org generates — unsigned, since the key does not exist yet. It is sent only once your consent is recorded, and only for an administrator who can record it: the server checks both before the call, and a refusal sends nothing;
- the **retrieval** of the HMAC key and of the viewer token: your org Id and that claim secret, which your org erases once the pair is stored. It is repeated until the publisher approves the request: by the screen while it stays open, for up to ten minutes at a time; by a background job, for up to about twelve hours, as long as conditions 1 to 3 below still hold; and again whenever an administrator reopens the screen while the request is still pending. Every retrieval, the screen's included, is sent only once your consent is recorded and only if the identity that sends it can store the pair: the server checks both before the call, and a refusal sends nothing. Once the pair is written, the key is read back: a pair that was not stored is reported as such instead of being announced stored.

Publication requires, **cumulatively**:

1. an **effective entitlement** — the **Heroku option** feature enabled on a sufficient licence plan, or a time-limited trial in progress;
2. a **persistent administrator consent**, stored in the **protected** custom setting `OrgGuardian_Heroku__c`, field `Consent_Granted__c`;
3. a **collector endpoint**, filled in on the **Heroku option** screen;
4. the **HMAC signing key** (field `Hmac_Key__c` of the same protected setting), retrieved from the collector once the publisher approves your activation request, or entered by your administrator.

The four conditions are checked again before each publication (Troubleshooting Guide §11).

Any error or any missing flag returns “no”: the locked-down posture prevails over doubt.

What activation opens. An outbound publication to an external viewing service, whose default endpoint is `https://viewer.orgguardian.com` — this is the **only** hard-coded external host in the package (the “Hard-coded external host” row of the table in §3). It remains overridable by the subscriber (`Endpoint__c`), and the value actually in force is shown in the **Heroku option** pane of the Settings tab:

that value is the one that is authoritative for your org, not the default value. What is published is an aggregated health summary — your org Id, the health score, finding counts by severity, a metric count, region and timestamp — never any finding content. Before it leaves, the payload passes the package's personal-data guard: a Salesforce user identifier (`005…`) or an email address anywhere in it, or a forbidden key such as a user, session or IP address field, aborts the send — fail-closed. The HMAC key that signs the payload never travels with it, and no user of your org can read it: it is held in the **protected** custom setting `OrgGuardian_Heroku__c` (field `Hmac_Key__c`), which only the package's code can read. When **Certificate and TLS expiry monitoring** is enabled as well, the org also sends the host names of its external endpoints to the same service, in a signed request, so that the service checks their TLS certificates.

The viewer link is readable by anyone who holds the full URL. Once the option is active, the **Heroku option** pane gives an administrator a **Viewer link (read-only)** to the Heroku dashboard of the org. The link carries its own access token: whoever holds the full URL can read that dashboard, so treat the link as a secret. The token is held in the same protected custom setting (field `Viewer_Token__c`) and is never displayed again after saving; the pane rebuilds the link only for an administrator, and only while publication is possible. The token is bound to the collector that issued it (`Issuing_Collector__c`: the address of the activation claim, or the one saved with a token entered by hand): the link is composed for that collector's host only. If the option's address moves to another host, no link is given, and the pane says so in place of the copy button, until an activation with the new collector stores a token it issued; pointing back to the original address restores the link, without asking the publisher for anything. A token with no recorded issuing collector (a token entered without an address) is bound to no host: no link is given, whatever the configured address, and the pane asks you to run the activation again with the configured collector. In the same cases (a token issued by a collector other than the configured one, or a token with no recorded issuing collector), the summary publication and the certificate probe no longer sign with the stored pair: publication is refused, nothing leaves the org, and the pane gives the reason, until an activation with the configured collector stores its own pair. A key entered alone, without a token, still signs.

The data flow of this option is described in detail in a dedicated document of the compliance dossier, available on request.

---

7. Data retention and deletion

The product purges **its own** records through a nightly batch (`OrgGuardian Data Retention`). Default windows, all adjustable by the subscriber in the **General settings** pane of the Settings tab:

Data	Default window	Setting key (`Org_Setting__mdt`)
Finding with no new observation	90 days	`Retention_Finding_Days`
Resolved finding and incident	30 days	`Retention_Resolved_Days`
Current metrics	30 days	`Retention_Metric_Days`

Data	Default window	Setting key (`Org_Setting__mdt`)
Technical logs	30 days (7 days on an org with 20 MB of data storage or less)	`Retention_Log_Days`
History snapshots	180 days	`Retention_Snapshot_Days`
Compliance evidence	400 days	`Retention_Compliance_Days`
Replayed predictions	400 days	`Retention_Prediction_Days`
Archived board reports	400 days	`Retention_Posture_Report_Days`

These eight windows are **deliberately configurable**: two subscribers subject to the same GDPR legitimately land on 30 and on 400 days. The 7-day default for technical logs applies only to an org whose data storage allowance is 20 MB or less (a Developer Edition, for one), and only while no value is recorded for `Retention_Log_Days`. Only one bound is not configurable — the cap of **7 days** that the `FREE` plan (the Discovery offer) applies to every window except the archived board reports, which keep their configured window on every plan: the cap has **no** `Org_Setting__mdt` key, so no setting raises it.

Uninstall. The uninstall script tries to cancel every scheduled job of the package (prefix `OrgGuardian`, visible in Setup → Scheduled Jobs) and **deletes nothing else**: it neither reads nor destroys any org data, and its read of its own scheduled jobs is itself in user mode. It never raises an error: when it cannot see or cancel a job, it says so only in the debug log of the uninstall. Measured on 2026-10-04 on one of the publisher's orgs, with an earlier version of the package (eleven scheduled jobs in place, permission sets unassigned, no Custom Metadata record written in the org): the uninstall was not refused, and no job remained after it, but the measurement does not show whether the script or the platform removed them. Even so, delete these jobs yourself before uninstalling (this does not depend on the script being able to see them), and check Setup → Scheduled Jobs again once the package is gone. The records in the package's objects follow the platform's standard behaviour for an uninstalled managed package. The Uninstall Guide describes the whole procedure in its §4, “The procedure” (these two steps are its §4.2 and §4.3), and what the uninstall destroys in its §1.

---

8. Where to find the rest

In this document set.

Question	Document
Prerequisites, installation and first scan	Installation Guide
Personas, permission sets and access	Permissions and Personas
No-code editors and feature activation	Configuration Guide
Scheduled jobs, findings lifecycle, alerts and exports	Operations Guide
Operating symptoms and degradations	Troubleshooting Guide
Vocabulary, objects and screens	Glossary

Question	Document
What the uninstall destroys	Uninstall Guide

Available on request: four documents from the publisher's compliance dossier go deeper into, respectively, the egress surface and credentials, the post-install exception, the data flow of the Heroku option and the WCAG 2.1 accessibility audit. Ask for them by name.