Troubleshooting Guide

Administrator guide — subscriber org. Each section gives the symptom, its cause, how to check it and how to fix it. The "What you can observe" paragraph says what you can see in your org — screen, message, record or query — to confirm the diagnosis.

Object and field names are given without the `orgguardian__` prefix that they carry in your org; queries use the full name.

Related documents: the Installation Guide (installation and first scan), the Permissions and Personas guide (permission sets and personas), the Security FAQ, the Glossary.

Table of faults

#	Observed symptom	Section
1	After installation, no scheduled job and no finding	§1
2	An administration command is refused: authorisation	§2
3	Same command, different refusal: the identity cannot write	§3
4	Finding "OrgGuardian could not read …" with severity High	§4
5	The Home screen cannot tell whether OrgGuardian is bootstrapped	§5
6	The Event Monitoring panels stay empty	§6
7	The screen responds, the feature returns nothing	§7
8	Partial results on a large org	§8
9	Asynchronous queue depth finding	§9
10	After uninstall, scheduled jobs remain	§10
11	Publishing to the external service stays silent	§11

1. After installation, the org stays empty

Symptom. The package is installed, no error has been reported, and yet: zero jobs in *Setup* → *Scheduled Jobs*, zero `Scan_Run__c`, zero `Finding__c`, zero `Org_Metric_Current__c`.

Cause. This is the product's normal behaviour, not a fault. The post-install script schedules nothing: it records an installation trace and stops there. It runs as a system user of the package, with neither a profile nor a permission set, whereas a scheduled job runs under the identity of the user who scheduled it. OrgGuardian therefore waits for a real user: its jobs are scheduled, and its first scan is started, the first time an administrator opens one of its screens.

Check. *Setup* → *Scheduled Jobs* lists no job whose name starts with `OrgGuardian`. From the command line:

```
sf data query --target-org <your-alias> --query \
  "SELECT COUNT() FROM CronTrigger WHERE CronJobDetail.Name LIKE 'OrgGuardian%'"
```

Fix. Open the **Home** tab of the OrgGuardian app as an administrator holding `OrgGuardian_Admin`: the jobs are scheduled and the first scan starts. Failing that, run the two administration commands of the package as anonymous Apex (Developer Console, or `sf apex run`):

```
System.debug(orgguardian.OrgGuardianAdmin.scheduleMonitoringJobs());
System.debug(orgguardian.OrgGuardianAdmin.runScanNow(false));
```

The first schedules any missing job and never creates a duplicate. The second queues one scan and returns a text that starts with `OK:`, `REFUSED:` or `ERROR:`.

Once complete, the bootstrap schedules **twelve** jobs: `OrgGuardian Hourly Scan`, `OrgGuardian Data Retention`, `OrgGuardian Compliance Snapshot`, `OrgGuardian Risk Resurface`, `OrgGuardian SLA Compliance`, `OrgGuardian Escalation Chains`, `OrgGuardian Endpoint Health Check`, `OrgGuardian Board Report`, `OrgGuardian Fleet Sync`, `OrgGuardian Posture Digest`, `OrgGuardian ITSM Sync`, `OrgGuardian Slack Reactions` — eleven maintenance jobs plus the hourly scan. Details in the Operations Guide §1.

What you can observe. Once the **Home** tab has been opened by an administrator holding `OrgGuardian_Admin`, *Setup* → *Scheduled Jobs* lists the twelve names above, and a first `Scan_Run__c` appears when the first scan completes. If the bootstrap fails, the Home screen says so in a banner, **OrgGuardian bootstrap did not complete**, which states what to check.

> **Good to know.** These twelve jobs are the complete list: there is no other OrgGuardian job to schedule by hand. The reading of Slack reactions is scheduled unconditionally and stays inert until its conditions are met: a paid plan or its trial, with the `Chat_Reactions` and `Channel_Slack` features switched on; the consent; the channel ID; the name of the bot's Named Credential; and, for the user who scheduled it, the access needed to record what it reads.

---

2. An administration command is refused: authorisation

Symptom. The first command of §1 stops with an error whose message begins “Driving OrgGuardian's scheduled jobs and scans requires…”, and **no** job is scheduled. Run on its own, the second returns `REFUSED:` followed by the same message.

Cause. The user running the command holds neither the custom permission `OrgGuardian_Manage_Features` (“OrgGuardian: manage features”) nor one of the platform permissions “Customize Application” or “Modify Metadata Through Metadata API Functions”. The command refuses **before** doing anything: no job is scheduled and nothing is recorded.

Check. Verify that the user holds `OrgGuardian_Admin`: of the five permission sets shipped with the package, it is the only one that carries `OrgGuardian_Manage_Features` (*Setup* → *Permission Sets* → ***OrgGuardian: technical administration*** *(`OrgGuardian_Admin`)* → *Custom Permissions*).

Fix. Assign `OrgGuardian_Admin` to the user who runs the command, or run it as a system administrator.

What you can observe. After the refusal, *Setup* → *Scheduled Jobs* is exactly as it was before the call: no job partially scheduled, no record created.

---

3. An administration command is refused: the identity cannot write

Symptom. Same command, **different message**: it begins "Your identity does carry the permission to drive OrgGuardian, but it lacks the write access the operation requires." Run on its own, the second command returns `REFUSED:` followed by the same message, and **no scan is queued**. On an org that is not yet bootstrapped, the Home screen shows the banner **OrgGuardian is not bootstrapped yet** for the same reason: the user's profile cannot create the findings and metrics that a scan produces.

Cause. The user may ask, but cannot write what a scan produces. A scheduled job runs under the identity of the user who scheduled it, so OrgGuardian schedules and launches nothing under an identity that could not record the results. This typically happens when a permission set of your own grants `OrgGuardian_Manage_Features` without the object permissions.

Check. The user needs **Read, Create and Edit** on both `Finding__c` and `Org_Metric_Current__c`: *Setup* → *Permission Sets* → *(the user's permission set)* → *Object Settings*.

Fix. Assign `OrgGuardian_Admin`, which grants these permissions, rather than a permission set that carries only the custom permission.

What you can observe. The message tells the two refusals apart: §2 concerns the right to ask, this section the ability to write. Once `OrgGuardian_Admin` is assigned, the same command goes through.

---

4. Finding "OrgGuardian could not read …": missing setup permission

Symptom. A finding appears in the list, category **Errors**, severity **High** (**Info** for `LoginGeo`, see below), with this message:

> OrgGuardian could not read *source* (insufficient access): the results may be incomplete.

where *source* names the object that could not be read. Users whose Salesforce language is French see this message in French on the product's screens; a report or a SOQL query returns the recorded English sentence. The finding's deduplication key starts with `OrgGuardian:ReadFailure:`.

Cause. The product reads several **setup** objects whose access depends on platform permissions that are not always granted. When a read is refused, the scan does not stop: it carries on and records this finding. The main sources, and what a refusal hides:

Refused source	What becomes invisible
`SetupAuditTrail`	The stream of setup changes over the last 7 days
`PermissionSetAssignment`, `User`	No new active human account that no permission forces to use MFA is reported, and those already reported stay open; the coverage rate and the account census (never logged in, dormant, frozen, federated) are recorded without a value, on a `User` refusal as on a `PermissionSetAssignment` refusal
`EventLogFile`; the Event Monitoring log objects (`EventLogObjects`)	The Event Monitoring panels — see §6

Other sources can appear: the finding's `Target__c` field always names the exact one. When the Monitoring pack's read of `ApexClass` or `ApexTrigger` is refused, the API version health score is not published, and the outdated API version finding stays open instead of closing. When the Code quality pack's read of the same objects is refused, the "Secure-by-Default Readiness" and "Next Salesforce Release Readiness" scores show no value, and the findings of those two detectors already open stay open. A **benign** refusal — the object simply does not exist in your org's edition — produces **no** finding. *(The "not supported" form is not read as benign for an object that exists in every org, such as `AsyncApexJob`, `ApexClass`, `ApexTrigger`, `User`, `PermissionSetAssignment`, `SetupAuditTrail` or `LoginHistory`, nor for the package's own objects: it is reported there as an access refusal, whichever pack reads the object. A source that some orgs do not expose, such as `EventLogFile`, `LoginGeo` or `ApexUnexpectedException`, keeps the benign reading, and so does an object that Salesforce does not always serve to the scan account, such as `ConnectedApplication`, `OAuthToken` or `PackageLicense`.)*

AsyncApexJob and LoginHistory. When reading one of these objects **fails** during an analysis — insufficient access, or an unexpected error — OrgGuardian opens a High finding in the Errors category: "OrgGuardian could not read AsyncApexJob (insufficient access): the results may be incomplete." The reason reads "(unexpected error)" when the failure is not an access refusal, and the object name reads LoginHistory for that object. While it is open, the finding is alerted like any High finding. A read diagnostic is not a control and weighs on no score, neither a pack score nor the health score, but the controls that the read feeds are credited with no pass as long as it is not complete.

The findings that this read feeds and that are already open **stay open** instead of being reported as fixed:

- for AsyncApexJob: failed or aborted jobs, queue depth (see §9), run time, plus the unhandled-exception findings, whose batch errors come from AsyncApexJob;
- for LoginHistory: the failed-login rate and the per-user brute-force findings.

While the read fails, the indicators it feeds (failed logins, failed jobs, queue depth, run time per job type) show no value ("—" or "Not measured", depending on the screen) instead of keeping the last value measured, and the trends add no point. The run time is then recorded without a value for every job type that Salesforce knows, including a type that never ran in your org; once the read is restored, a type that

has not run in the last 24 hours keeps that row without a value until its next run, at most the retention period of the indicators (30 days by default, 7 days on the free plan). The unhandled exceptions count shows no value either while the read fails, rather than the count of the only source still readable.

Three limits apply.

1. **On LoginHistory, a missing permission does not make the read fail: it filters it.** Salesforce then returns only the logins of the user who runs the analysis. OrgGuardian detects it: when that user holds neither "Monitor Login History" nor "Manage Users", the read is not made, and the finding above opens with the reason "(insufficient access)", with the same effects as a refusal; the breakdown of failed logins by reason and by type, and that of logins by country, are then recorded without a value instead of keeping the last one (if the post-collection reserve allows it). If OrgGuardian cannot read that user's permissions, it does not read the history either, and the finding opens with the reason "(unexpected error)". For least privilege, schedule the analysis as a user who holds "Monitor Login History" rather than "Manage Users", which is far broader: it is enough for the failed-login rates and the per-user breakdown. It does not cover `LoginGeo`, which Salesforce reserves to "Manage Users": without that permission, OrgGuardian does not read the geographic enrichment of logins, and an **Info** finding, "could not read LoginGeo (insufficient access)", signals it: under that permission, the by-country anomalies are not measured. It stays open as long as the permission is missing; it lowers no score, neither a pack score nor the health score: it is not a control. The logins-by-country breakdown is recorded without a value instead of keeping the last one (if the post-collection reserve allows it), and the "new country" and "impossible travel" anomalies already open stay open.
2. The same LoginHistory finding is also opened by the API exfiltration detector (Security pack, on by default on a paid plan) and by the login anomaly detection (Security pack, off by default) when their own read fails or is filtered; their findings already open stay open. It stays open as long as any detector reports the failure.
3. The finding closes by itself at the end of the first analysis in which the object is read successfully and no pack or analysis step that reads it failed (see **Fix** below).

Above 5,000 logins in 24 hours, the per-user breakdown is suspended (a Medium finding says so), and the per-user brute-force findings already open stay open until the breakdown runs again.

Large reads are capped. When one is cut short, an **Info** notice says so. An Info notice lowers no score, neither a pack score nor the health score: it is not a control and makes no control fail. A partial read does, however, remove from the evaluated set the controls it did not read in full: the pack score and the health score count only what was read, and they can move without any fix when that coverage changes (the displayed change is then marked "Coverage changed"). Five of these capped reads bear on the sources of this section:

- above 2,000 async jobs created in the last 24 hours and completed, only the 2,000 most recently completed are read. The average run times cover those jobs only. The run-time findings already open for the job types absent from those 2,000 stay open until a run reads that type again or reads the whole window; those of a job type present close when its average run time falls back below the threshold.
- when the inventory of your own Apex classes and triggers is read only in part (row budget exhausted, or cap of 20,000 rows reached), the API version health score is not published, and the outdated API version finding stays open.

- above 2,000 distinct groups of unhandled exceptions or batch errors over 24 hours, for either of the two sources, only 2,000 groups of that source are read: the unhandled exceptions count published covers those only, and the unhandled-exception findings already open stay open.
- when the Code quality pack reads the inventory of your own Apex classes and triggers only in part (above 5,000 classes or 5,000 triggers, or row budget exhausted), the "Secure-by-Default Readiness" and "Next Salesforce Release Readiness" scores show no value, and the findings of those two detectors already open stay open, including the finding of a class read and fixed in the meantime, which closes at the first complete analysis.
- above 5,000 distinct (client, status, user) combinations over the last 7 days, or over the 23 days before them, the "API clients" slice of the API exfiltration detector (Security pack, on by default on a paid plan) emits no client finding, and its findings already open stay open.

The per-user breakdown of failed logins reads every login of the last 24 hours, up to 5,000, the very threshold above which it is suspended. If it is cut short anyway, the finding "could not read LoginHistory (unexpected error)" says so, and the per-user findings already open stay open.

One read degrades **without** this finding: `CronTrigger` (the bootstrap state on the Home screen, see §5). `LoginGeo` is silent too when the object does not exist in your edition: there is nothing to read then.

Check. Verify the corresponding platform permission for the user **who scheduled** the jobs — the jobs run under that user's identity: the permission that gives access to the setup audit trail, “View Event Log Files”, “View Event Log Object Data”, “View Setup and Configuration”, “View All Users”; for LoginHistory, “Monitor Login History” or “Manage Users”; for LoginGeo, “Manage Users”. Before each analysis, OrgGuardian also checks the four read rights that the scan account needs — “View Setup and Configuration”, “View Roles and Role Hierarchy”, “View All Users” and “View All Profiles” — and names each missing one in a High notice of the Security category, whose key starts with `OrgGuardian:ScanRights:`. See the Installation Guide §5.1 bis.

Who receives which permission. The read-only ones (“View Event Log Files”, “Monitor Login History”, and the four of the `OrgGuardian:ScanRights` notices) go to the **scan account** and to the users who click **Run scan now**, which reads with the rights of the user who clicks. “Manage Users”, which only `LoginGeo` requires, also lets its holder reset passwords and assign permission sets: weigh it, then grant it to the scan account only, never to every user who clicks **Run scan now**. A **Run scan now** launched without it leaves the geographic checks unread: the finding names `LoginGeo`, and the next scheduled scan, run by the scan account, reads it.

Fix. Grant the permission, then wait for the next scan. The finding closes **on its own** once the read succeeds again, at the end of the scan, provided that no pack or analysis step that reads its source failed during that scan. A pack or step that fails leaves open the findings of the sources it reads, which wait for the next scan. When the only pack that reads the source does not run because your plan does not include it — or, on the free plan, runs only its free security core — the finding closes without a verification date, even when an analysis step such as Health Check has read the source again: no fix is claimed, and if it comes back it is not counted as a regression. In that case, only a detector of that scan that attests it read the source in full gives the closure a date. No pack closes its own read findings: only the end of the scan does. The health score has no diagnostic of its own: it is computed with the pack scores, and a refused read of the findings opens the `Finding__c (PostureScores)` diagnostic, which leaves all these values, health score included, at those of the previous scan. There is no manual “Resolved” action for these findings.

What you can observe. These findings can be listed from your org:

```
sf data query --target-org <your-alias> --query \
  "SELECT orgguardian__Target__c, orgguardian__Severity__c \
  FROM orgguardian__Finding__c \
  WHERE orgguardian__Dedupe_Key__c LIKE 'OrgGuardian:ReadFailure:%'"
```

Each row names its source in `Target__c`. The scan of the same cycle has nonetheless completed, and the other findings have been written: a scan that meets a refused read carries on, and for these sources it says so.

---

5. The Home screen cannot tell whether OrgGuardian is bootstrapped

Symptom. The Home screen shows the banner **Bootstrap state cannot be verified with your profile**: it neither confirms nor denies that OrgGuardian is bootstrapped.

Cause. The user's profile cannot read the list of scheduled jobs, which belongs to the org configuration. Rather than guess, the screen says that it cannot tell. The other banners that the Home screen can show are **OrgGuardian is starting up** (first scan in progress), **OrgGuardian is not bootstrapped yet** (an administrator must open the screen, see §3) and **OrgGuardian bootstrap did not complete** (the jobs could not be created; the banner says what to check). When everything is in place, no banner is shown.

Check. Open the same screen as an administrator holding `OrgGuardian_Admin`: the exact state is displayed, and the bootstrap completes if anything is missing.

Fix. No action on the product is required. If the display must be conclusive for this user, grant them “View Setup and Configuration”.

What you can observe. As an administrator, *Setup → Scheduled Jobs* lists the twelve jobs of §1. For a read-only user, the banner disappears once a scan has completed: the results they can see then show that OrgGuardian is running.

---

6. Empty Event Monitoring panels

Symptom. The screens fed by Event Monitoring display nothing, with no error message.

Cause. Three distinct causes, not to be confused.

1. **The org does not have Event Monitoring.** OrgGuardian works **without** it: Event Monitoring *enriches* the collection; it is not a precondition for it. The product looks for (a) at least one of three Event Monitoring log objects readable by the user — `LightningPageViewEventLog`, `ApexUnexpectedExcpEventLog`, `LightningErrorEventLog` — and (b) the paid event types `LightningPageView`, `LightningPerformance` and `ApexExecution` in `EventLogFile`. With neither, the derived panels stay empty, and the rest of the product works normally.
2. **The read permission is missing.** The panels stay empty in the same way, and a finding "OrgGuardian could not read `EventLogFile`" (or `EventLogObjects`) appears — see §4.
3. **The read of the log files is switched off, or not wired.** OrgGuardian reads the content of the log files through a call from your org to itself, using the Named Credential shipped with the package. That Named Credential and its External Credential ship unwired: the address is a placeholder that does not resolve, and the credential is empty, until your administrator wires them, once. The read is one of the outbound features: its switch, **Event Monitoring ELF fetch**, is off by default, and your plan must include it (see §7). While it is off, the panels derived from the log files stay empty, with no finding and no error. Switched on before the wiring, the read fails closed: nothing is read, and the panels stay empty.

Check. Look for the finding of §4 first: if it is present, the cause is 2. Otherwise, ask the platform directly:

```
# (a) the log objects: one of the three named above is enough
sf data query --target-org <your-alias> \
  --query "SELECT Id FROM LightningPageViewEventLog LIMIT 1"
# (b) the event types present in EventLogFile
sf data query --target-org <your-alias> \
  --query "SELECT EventType FROM EventLogFile GROUP BY EventType"
```

An **unknown entity** error means that the org does not have the feature (cause 1). A **permission** error means that the feature is there and the permission is missing (cause 2). A result with no error, empty or not, means that the feature and the permission are both there: then check the **Event Monitoring ELF fetch** switch in the **Feature activation** section, and the wiring of the self-callout (cause 3). If both are in place and the result is empty, there is simply nothing to read yet.

Fix. Cause 1: nothing to fix in OrgGuardian — subscribe to Event Monitoring if you need these panels. Cause 2: grant the user who runs the scheduled jobs "View Event Log Files" (finding on `EventLogFile`) or "View Event Log Object Data" (finding on `EventLogObjects`). Cause 3: wire the self-callout once (the Configuration Guide §3, "Wiring the `OS_SelfOrg` self-callout"), then turn on **Event Monitoring ELF fetch**, on a plan that includes it, from the **Tooling API access** pane or in the **Feature activation** section.

What you can observe. In every case the rest of the product keeps running: scans complete and the other panels fill up.

---

7. The screen responds, but the feature returns nothing: licence plan

Symptom. The screen displays, reports no error, and the panel stays empty or the rule never applies.

Cause. The feature depends on the **plan**, and the org is on the free plan (Discovery, technical tier `FREE`). An org that the publisher has not provisioned, and whose 30-day trial of the full plan is not running — never started, or over — is on the free plan, and every feature reserved to a paid plan stays closed. The package ships no licence record. The free plan still runs a free security core: human accounts without MFA, guest user exposure, public file links, connected apps and OAuth tokens, and the holders of Modify All Data and View All Data. The rest of the Security pack needs a paid plan.

The order is fixed: the plan provisioned by the publisher applies; failing that, a 30-day trial of the full plan that is still running — it starts the first time an authorised administrator opens an OrgGuardian screen; failing that, and whatever the reason — absent, unreadable, read error — the free plan.

Two **independent** locks, not to be confused:

- **Plan** — what the org is **entitled** to use;
- **Feature switch** — what actually **runs**. Every outbound feature is off by default, whatever the plan.

A sufficient plan therefore does not, on its own, open any outbound flow.

Check. The **Subscription** screen (App Launcher → **Subscription**) displays the plan: “OrgGuardian, paid plan”, “Paid-plan trial” or “OrgGuardian Discovery, free plan”. The **Feature activation** section (Administration ▸ Settings ▸ Feature activation) displays the switch of each feature.

Fix. The plan is **provisioned by the publisher**, never by the subscriber: request it from the publisher. A feature's switch, on the other hand, is yours to turn on, in the **Feature activation** section.

What you can observe. The case to recognise is the one where the two seem to contradict each other — paid plan displayed, panel still empty: the feature's switch is off, and no plan turns it on. Conversely, on the free plan, a feature reserved to paid plans is shown with a padlock and its switch cannot be turned on; a switch turned on during the 30-day trial no longer makes the feature run once the trial is over. Both are needed: the plan, set by the publisher, and the switch, set by you.

A feature that is turned off is never a verified fix. When a feature's switch is off, the feature no longer runs, and the next scan closes its findings that are still open **without a verification date**. They leave the score. They count neither in "Verified fixed" on the **Remediation effectiveness** tile of the **Findings** screen, nor in "Fixes verified by a re-scan of the product" on the **Value ledger** screen, nor in the number of resolved findings of the posture digest, and their row on the **Findings** screen does not show "Verified fixed on". When the feature is turned back on, a finding that returns is not counted as a regression.

- **What still counts them.** Two lines of the **Value ledger** count every closure, these ones included: "Findings closed by your teams" and "Critical findings closed by your teams". They are raw counts of closures, not proofs of a fix.

- **The same rule, wherever the switch is read.** Public groups, Event Monitoring data freshness and backup posture close their findings without a date when their feature is off. So do two sub-detections read in the middle of a feature that is on, when their own switch is off: the API clients of the exfiltration detection, and the org hardening checks that only the Tooling read evaluates.
- **Kept open: last known state, never a fix.** With their feature off, these capabilities keep their findings open, without a date, for as long as they stay off: CORS, authentication surface, identity verification, orphaned scheduled jobs, page experience, custom rules, intelligence, connectivity probe, endpoint registry, SLA, and deviations from the approved baseline.
- **One switch, two readings.** "Tooling API discovery scanners" is the switch of the Tooling scanners of outbound calls: off, their findings close without a date. For the Health Check and the declarative linter it is the permission to read: off, they report themselves as not measured and keep their findings open.
- **Still closed with a date.** An unhandled-exception fingerprint that only the Event Monitoring read saw, when Event Monitoring is off: it carries the same key as those of the free exception detector.

---

8. Partial results: query row budget reached

Symptom. On a large org, a pack returns fewer findings than usual, and an Info finding reports a truncation.

Cause. A Salesforce transaction may read at most **50,000 rows**; going over that limit stops the transaction with an error that cannot be caught, and all the findings of the run would be lost. The reads of a pack therefore share a common budget within its transaction. When the budget runs out, the remaining reads are capped or skipped, and the scan declares it in an Info finding instead of guessing: existing findings are kept, and nothing is resolved on the strength of a partial read.

Check. The truncation finding names the analysis concerned. The rest of the run is written normally.

Fix. No action is needed for an occasional truncation: the next scan starts again with a fresh budget. If it recurs on every scan, contact support at orgguardian.com/en/support.html.

What you can observe. The truncation is declared, never silent. The scan run has completed, the other analyses have written their findings, and **no existing finding has been resolved in bulk**: the number of open findings does not collapse after a truncated scan.

A read notice — truncation, unavailable inventory, refused or unmeasured read, abstention of a detector, missing observation by the product itself (scheduled scan stopped, incomplete Event Monitoring window, gaps in the product's jurisdiction or ITSM mapping table) — closes without a verification date as soon as it is no longer emitted: the read is complete or possible again, or the detector can conclude again. It counts neither in "Verified fixed" on the **Remediation effectiveness** tile of the **Findings** screen, nor in "Fixes verified by a re-scan of the product" on the **Value ledger** screen, nor in the number of resolved findings of the posture digest, and its return is not counted as a regression; its row on the **Findings** screen shows neither "Verified fixed on" nor the "Came back" regression badge. Open or closed, a notice is not counted either in "Risks detected and investigated by the product" on the **Value ledger** screen: it says that a read was bounded or impossible, not that a risk was detected.

---

9. Saturated asynchronous queue (FlexQueue)

Symptom. A finding reports that the asynchronous queue depth exceeds the threshold; scheduled scans start late.

Cause. OrgGuardian counts the asynchronous jobs that are waiting or in progress, and raises a finding above a threshold set by default to **50**. The cause is almost always **outside the product**: another process in the org saturates the queue, and the OrgGuardian jobs wait their turn.

Check.

```
sf data query --target-org <your-alias> --query \
  "SELECT COUNT() FROM AsyncApexJob \
  WHERE Status IN ('Queued','Holding','Preparing','Processing')"
```

Fix. Spread out the competing asynchronous processes.

What you can observe. The finding's message gives the number of jobs counted and the threshold; the query above counts the same statuses at the moment you run it. If `AsyncApexJob` cannot be read by the user who runs the scheduled jobs, no queue depth finding is produced: the read-failure finding "OrgGuardian could not read AsyncApexJob" says so (§4), and a queue depth finding already open stays open instead of being reported as fixed.

---

10. Orphaned scheduled jobs after uninstall

Symptom. The package has been uninstalled, but scheduled jobs whose name starts with `OrgGuardian` are still listed.

Cause. The package ships an uninstall script that tries to cancel the scheduled jobs whose name starts with `OrgGuardian` — the twelve jobs of §1 — and touches nothing else. Measured on 2026-10-04 on one of the publisher's orgs, with an earlier build of the package whose uninstall script works the same way, in its default installation state (its scheduled jobs in place, permission sets unassigned, no Custom Metadata record written in the org): the uninstall was **not refused**, and **no** `OrgGuardian` job remained after it. The measurement does not show whether the uninstall script or the platform removed them. The script never raises an error: when it cannot read or cancel a job, it says so only in the debug log of the uninstall.

Check. *Setup* → *Scheduled Jobs*, filtered on `OrgGuardian`.

Fix. Delete the remaining jobs by hand. To avoid the case, delete them before uninstalling (the Uninstall Guide §4.2): it costs nothing, and it does not depend on the script being able to read `CronTrigger`. No customer data is affected: the script reads and deletes nothing other than the package's scheduled jobs.

What you can observe. Only jobs whose name starts with `OrgGuardian` are concerned: no job of another package, and none of your records, is touched by the script.

---

11. Publishing to the external service stays silent

Symptom. The Heroku option seems to be configured, but nothing is published: no outbound call, no error, and the screen reports no successful publication.

Cause. Publishing does nothing by default, and stays that way as long as a condition is missing. It requires, **cumulatively**:

1. **effective eligibility** — a sufficient plan with the Heroku option's switch turned on (**Feature activation**), or a **30-day** trial of the option still running — and, in both cases, the option not muted by an administrator;
2. **administrator consent**, given on the **Heroku option** screen;
3. **a collector endpoint** filled in on the same screen;
4. **the HMAC signing key**, retrieved from the collector once the publisher approves your activation request, or entered by your administrator.

The four conditions are checked again just before each publication, and nothing is kept from a previous check. A read error counts as a missing condition: no outbound call is then made.

A special case, not to be confused with a fault: the screen may show “Publish log unreadable: it could not be read (object inaccessible or permission missing). No conclusion can be drawn about the last publish.” The current user cannot read the publication log; this says nothing about whether publications take place.

Check. **Administration ▸ Settings ▸ Heroku option**, then expand **Advanced settings**: the checklist “Data-egress configuration state (what is set up, not what has been sent)” shows, item by item, “Feature switch enabled (Feature activation section)”, “Explicit consent granted”, “Collector endpoint set” and “HMAC key saved”. The key signs each publication: without it, no publication is sent. The consent, the endpoint and the key can be read on this screen only: they are kept in a protected setting that code in your org cannot read.

Fix. Go through the four conditions in order: plan and switch, or trial; then consent; then endpoint; then the key. A single missing one is enough to stop publication.

What you can observe. As long as eligibility or consent is missing, no publication is attempted, and nothing records the fact: no finding, no error, no log entry — which is why this section exists. Once both are in place, each scan ends with a publication attempt; if the endpoint or the key is missing, that attempt stops before any outbound call, and the **Heroku option** screen reports “Last publish: not emitted”, followed by the reason (for a missing key: “Aborted: HMAC signing key unavailable (protected secret not resolved).”). The attempt also stops before any outbound call when the key stored in your org came with a viewer token that is not bound to the collector now configured (issued by another collector, or with no issuing collector recorded): the reason then starts with “Aborted: the viewer token stored in this org is not bound to the collector now configured”; running the activation for this collector gives your org its own key and token. Withdrawing consent takes effect at the next publication, without a restart or rescheduling. When the 30-day trial ends, an org without a sufficient plan stops publishing without anything else having changed.

Outside the scope of this guide

- What the Heroku option changes **when it is switched on**, and the details of the payload emitted: the Security FAQ §6.
- The vocabulary of the objects and screens cited here: the Glossary.